PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2001-142918
(43)Date of publication of application : 25.05.2001

(51)Int.Cl. G06F 17/50
H01L 21/82
H01L 27/04
H01L 21/822

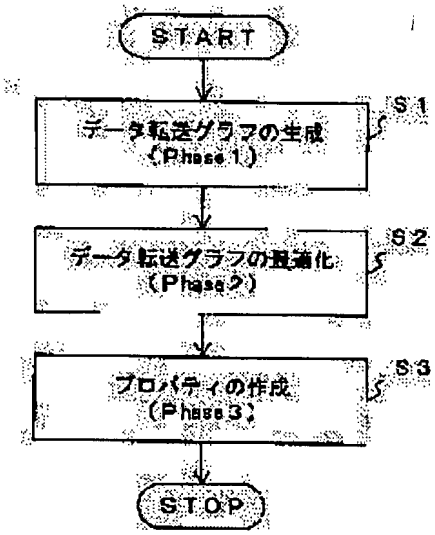
(21)Application number : 11-321346
(22)Date of filing : 11.11.1999

(71)Applicant : FUJITSU LTD
(72)Inventor : SUBIRU ROI
IWASHITA HIROAKI
NAKADA TSUNEO

(54) DEVICE AND METHOD FOR GENERATING PROPERTY TO BE VERIFIED ABOUT HARDWARE

(57)Abstract:
PROBLEM TO BE SOLVED: To provide a method for automatically generating a property to be verified about hardware described in a hardware description language.
SOLUTION: Data resources and a data transfer described in a hardware description language are expressed using a data transfer display graph (step S1). The data transfer graph is optimized (step S2). The data transfer graph is simplified or minimized by optimizing processing. On the basis of the topology of the optimized data transfer graph, the property to be verified by a verifying tool is generated (step S3). The property to be verified is resource contension and register omission.

プロパティ生成ツールの
動作を説明するフローチャート



LEGAL STATUS

[Date of request for examination]	13.11.2003
[Date of sending the examiner's decision of rejection]	
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]	
[Date of final disposal for application]	
[Patent number]	3943299
[Date of registration]	13.04.2007
[Number of appeal against examiner's decision of rejection]	
[Date of requesting appeal against examiner's decision of rejection]	
[Date of extinction of right]	

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（A）　(11)特許出願公開番号
特開2001－142918
（P2001－142918A）
(43)公開日　平成13年５月25日(2001.5.25)

(51)Int.Cl.'	識別記号	ＦＩ		テーマコード'（参考）
Ｇ０６Ｆ　17/50		Ｇ０６Ｆ　15/60	６７０Ｚ	５Ｂ０４６
Ｈ０１Ｌ　21/82			６６４Ａ	５Ｆ０３８
27/04			６７０Ｋ	５Ｆ０６４
21/822		Ｈ０１Ｌ　21/82	Ｔ	
			Ｃ	

審査請求　未請求　請求項の数17　ＯＬ　（全 41 頁）　最終頁に続く

(21)出願番号　特願平11－321348
(22)出願日　平成11年11月11日(1999.11.11)

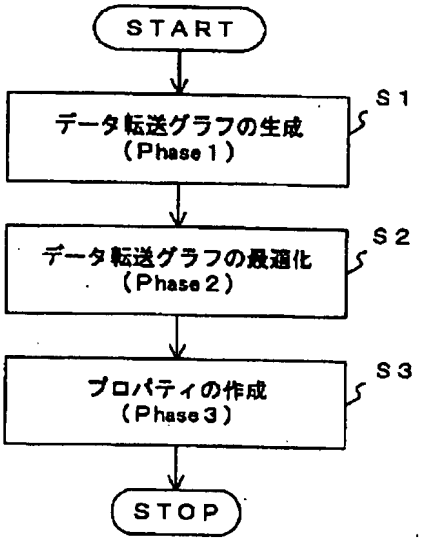
(71)出願人　000005223
富士通株式会社
神奈川県川崎市中原区上小田中４丁目１番１号
(72)発明者　スピル　ロイ
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内
(72)発明者　岩下　洋哲
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内
(74)代理人　100074099
弁理士　大菅　義之　（外１名）

最終頁に続く

(54)【発明の名称】　ハードウェアについて検証すべきプロパティを生成する装置および方法

(57)【要約】
【課題】　ハードウェア記述言語により記述されているハードウェアについて検証すべきプロパティを自動的に生成する方法を提供する。
【解決手段】　ハードウェア記述言語により記述されているデータ資源およびデータ転送をデータ転送表示グラフを用いて表す（ステップＳ１）。データ転送グラフを最適化する（ステップＳ２）。最適化処理により、データ転送グラフは簡略化または最小化される。最適化されたデータ転送グラフのトポロジに基づいて検証ツールにより検証すべきプロパティを生成する（ステップＳ３）。検証すべきプロパティは、資源競合およびレジスタ漏れである。

プロパティ生成ツールの
動作を説明するフローチャート



1

【特許請求の範囲】

【請求項１】　ハードウェア記述言語により記述された複数の資源およびそれらの資源を利用したデータ転送を含むハードウェアについて検証すべきプロパティを生成する装置であって、

上記ハードウェア記述言語で記述された資源およびデータ転送に対応するデータ転送グラフを生成するグラフ生成手段と、

そのグラフ生成手段により生成されたデータ転送グラフを最適化する最適化手段と、

その最適化手段により最適化されたデータ転送グラフを利用して、上記ハードウェア記述言語により記述されているハードウェアについて検証すべきプロパティを生成するプロパティ生成手段と、

を有するプロパティ生成装置。

【請求項２】　請求項１に記載のプロパティ生成装置であって、

上記最適化手段は、データ転送グラフのトポロジに基づいてそのグラフを最小化する。

【請求項３】　請求項１に記載のプロパティ生成装置であって、

上記グラフ生成手段は、各資源にそれぞれ対応するノード、および各データ転送の転送元および転送先を表すエッジを用いて上記ハードウェアを表すデータ転送グラフを生成する。

【請求項４】　請求項３に記載のプロパティ生成装置であって、

上記最適化手段は、互いに隣接する２つのノードに対応する各資源のビット幅が互いに同じであり、且つ一方の資源が他方の資源の子供である場合、それらのノードのうちの一方を上記データ転送グラフから削除する。

【請求項５】　請求項３に記載のプロパティ生成装置であって、

上記最適化手段は、第１の資源から第２の資源および第３資源にデータが転送され、それら第２および第３の資源のビット幅が上記第１の資源のビット幅よりも小さい場合、上記第１の資源に対応するノードを上記データ転送グラフから削除する。

【請求項６】　請求項３に記載のプロパティ生成装置であって、

上記最適化手段は、第１の資源および第２の資源から第３資源にデータが転送され、且つ上記第１および第２の資源のビット幅が上記第３の資源のビット幅よりも小さい場合、上記第３の資源に対応するノードを上記データ転送グラフから削除する。

【請求項７】　請求項３に記載のプロパティ生成装置であって、

上記最適化手段は、レジスタからそのレジスタの名称と同じ名称が付与されている出力ポートにデータが転送される場合、上記出力ポートに対応するノードを上記データ転送グラフから削除する。

2

【請求項８】　請求項３に記載のプロパティ生成装置であって、

上記最適化手段は、第１の資源から送出されるデータが第２の資源のみに転送され、且つ上記第２の資源が上記第１の資源から転送されてくるデータのみを受け取る場合、上記第１の資源から上記第２の資源へのデータ転送を表すエッジを上記データ転送グラフから削除する。

【請求項９】　請求項３に記載のプロパティ生成装置であって、

上記最適化手段は、第１の資源から第２の資源へのデータ転送と第１の資源から第３資源へのデータ転送との間で転送すべきビットがオーバラップしない場合に、上記２つのデータ転送に対応するエッジを上記データ転送グラフから削除する。

【請求項１０】　請求項３に記載のプロパティ生成装置であって、

上記最適化手段は、第１の資源から第２の資源へのデータ転送と第３の資源から第２資源へのデータ転送との間で転送すべきビットがオーバラップしない場合に、上記２つのデータ転送に対応するエッジを上記データ転送グラフから削除する。

【請求項１１】　請求項３に記載のプロパティ生成装置であって、

上記最適化手段は、複数のレジスタが同一のレジスタファイルに属する場合、それら複数のレジスタに対応する複数のノードを１つのノードに変換する。

【請求項１２】　請求項１に記載のプロパティ生成装置であって、

上記プロパティ生成手段は、資源競合およびレジスタ漏れのうちの少なくとも１つに係わるプロパティを生成する。

【請求項１３】　請求項１に記載のプロパティ生成装置であって、

上記プロパティ生成手段は、あるノードに複数のファンインエッジが接続されている場合、それら複数のファンインエッジに対応するデータ転送に対してそれぞれ与えられている条件に基づいて資源競合プロパティを生成する。

【請求項１４】　請求項１に記載のプロパティ生成装置であって、

上記プロパティ生成手段は、あるノードにファンインエッジおよびファンアウトエッジが接続されている場合、それらのエッジに対応するデータ転送に対してそれぞれ与えられている条件に基づいてレジスタ漏れプロパティを生成する。

【請求項１５】　ハードウェア記述言語により記述された複数の資源およびそれらの資源を利用したデータ転送を含むハードウェアについて検証すべきプロパティを生成する装置であって、

上記ハードウェア記述言語で記述された資源およびデータ転送に対応するデータ転送グラフを生成するグラフ生成手段と、
上記データ転送グラフのトポロジを利用して、上記ハードウェア記述言語により記述されているハードウェアについて検証すべきプロパティを生成するプロパティ生成手段と、
を有するプロパティ生成装置。

【請求項１６】　ハードウェア記述言語により記述された複数の資源およびそれらの資源を利用したデータ転送を含むハードウェアについて検証すべきプロパティを生成する方法であって、
上記ハードウェア記述言語で記述された資源およびデータ転送に対応するデータ転送グラフを生成し、
そのデータ転送グラフを最適化し、
その最適化されたデータ転送グラフを利用して、上記ハードウェア記述言語により記述されているハードウェアについて検証すべきプロパティを生成するプロパティ生成方法。

【請求項１７】　ハードウェア記述言語により記述された複数の資源およびそれらの資源を利用したデータ転送を含むハードウェアについて検証すべきプロパティを生成するためのプログラムを格納した記録媒体であって、
上記プログラムがコンピュータにより実行されたときに、
上記ハードウェア記述言語で記述された資源およびデータ転送に対応するデータ転送グラフを生成する手段と、
そのデータ転送グラフを最適化する手段と、
その最適化されたデータ転送グラフを利用して、上記ハードウェア記述言語により記述されているハードウェアについて検証すべきプロパティを生成する手段とを提供する記録媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、ハードウェア記述言語により記述された複数の資源およびそれらの資源を利用したデータ転送を含むハードウェアについて検証すべきプロパティを生成する装置および方法に係わる。

【０００２】

【従来の技術】近年、半導体チップの設計は非常に複雑になってきている。すなわち、半導体チップ上に非常に複雑なハードウェア（論理装置）が実装されるようになってきている。このため、半導体チップの設計は、複数の設計者による共同作業で行われることが多い。

【０００３】ところが、複数の設計者により細分化された設計情報は、各設計者により誤って解釈されることがある。この場合、ある一部における誤り（設計ミス）は、レジスタ転送レベルにおいて、システム全体が正しく動作しなくなる原因となることがある。そして、もし、そのような誤りがあると、デバッグ作業などに多大な費用と時間が浪費されてしまう。

【０００４】ところで、半導体チップ上に設けることができる資源（例えば、レジスタ、バス、入力ポート、出力ポートなど）は有限である。このため、各資源は、通常、複数のタスクにより共有される。そして、共有されている資源により生成されるデータは、しばしば、一時的にデータ記憶用資源（例えば、レジスタなど）に格納される。

【０００５】このような状況において頻繁に発生する設計誤りは、「資源競合」および「レジスタ漏れ」である。資源競合は、１つの資源（レジスタなど）が複数の資源により同時にアクセスされる誤りである。一方、レジスタ漏れは、複数の資源によりあるレジスタがアクセスされる際のアクセス順番に係わる誤りである。

【０００６】複雑なハードウェアが実装された半導体チップ上には、膨大な数の資源が形成されている。このため、上述のような誤り（資源競合、レジスタ漏れなど）を人手で全て発見することは実質的に不可能である。ところが、設計されたハードウェアの中にたった１つの誤りがあっただけでシステム全体が正しく動作しなくなってしまうこともある。したがって、すべての資源競合およびレジスタ漏れを発見して取り除いておくことは、非常に重要である。

【０００７】

【発明が解決しようとする課題】多くの設計者は、設計作業の早い段階で資源競合やレジスタ漏れを発見することの重要性を認識している。しかし、現在までのところ、資源競合やレジスタ漏れを発見するための作業は、各設計者によりその設計者が担当するサブシステムごとに行われているに過ぎなかった。

【０００８】システム全体の動作を保証するためには、サブシステムごとの検証だけでは十分ではなく、設計されたハードウェア全体に渡って資源競合やレジスタ漏れを探し、それらを取り除く必要がある。しかし、従来は、そのような作業はなされていなかった。このため、ハードウェア設計に誤りが含まれていた場合には、それらの誤りは、通常、半導体チップ上にそのハードウェア設計を実装した後に検出されることになる。

【０００９】本特許出願の出願人が知る範囲では、資源競合およびレジスタ漏れを自動的に検出し、定式化し、確認するツールは存在しない。そして、そのようなツール無しでは、誤りのないハードウェア設計を実現することは実質的に不可能であろうと思われる。

【００１０】本発明の課題は、ハードウェア記述言語により記述されているハードウェアについて検証すべきプロパティを生成することができる装置および方法を提供することである。

【００１１】

【課題を解決するための手段】本発明のプロパティ生成装置は、ハードウェア記述言語により記述された複数の

資源およびそれらの資源を利用したデータ転送を含むハードウェアについて検証すべきプロパティを生成する構成を前提とし、以下の各手段を有する。グラフ生成手段は、ハードウェア記述言語で記述された資源およびデータ転送に対応するデータ転送グラフを生成する。最適化手段は、そのグラフ生成手段により生成されたデータ転送グラフを最適化する。プロパティ生成手段は、その最適化手段により最適化されたデータ転送グラフを利用して、上記ハードウェア記述言語により記述されているハードウェアについて検証すべきプロパティを生成する。

【００１２】上記構成において、ハードウェア記述言語により記述されている設計情報がグラフ化され、そのグラフを利用してハードウェアについて検証すべきプロパティが自動的に生成される。このとき、プロパティが生成される前にデータ転送グラフが最適化されるので、プロパティの生成が容易になる。生成されたプロパティは、検証ツールにより検証される。

【００１３】資源は、例えば、レジスタ、バス、入力ポート、出力ポートであり、データ転送グラフにおいてそれぞれノードで表される。一方、データ転送は、その転送元および転送先が定義されたエッジ（矢印）を用いて表される。そして、最適化手段は、データ転送グラフのトポロジに基づいてグラフを最小化する。

【００１４】生成すべきプロパティは、例えば、資源競合およびレジスタ漏れである。資源競合のプロパティは、データ転送グラフにおいて、複数のファンインエッジが接続されたノードにおいて検出される。一方、レジスタ漏れは、データ転送グラフにおいて、ファンインエッジおよびファンアウトエッジが接続されたレジスタを表すノードにおいて検出される。

【００１５】

【発明の実施の形態】以下、本発明の実施形態について図面を参照しながら説明する。図１は、本発明の一実施形態のプロパティ生成ツールが使用される環境を説明する図である。本実施形態のプロパティ生成ツール１は、予め作成されているハードウェア仕様２からプロパティを抽出し、それを検証ツール３に検証させるものである。

【００１６】ハードウェア仕様２は、あるハードウェア（例えば、半導体チップ上に実装されるプロセッサ等の論理装置）の設計記述であり、そのハードウェアに含まれる複数の資源、およびそれら資源間のデータ転送を定義する。このハードウェア仕様２は、たとえば、レジスタ転送レベルのハードウェア記述言語で記述されている。ハードウェア記述言語は、例えば、Verilog である。また、定義される資源は、例えば、レジスタ、レジスタファイル、バス、入力ポート、出力ポート、入出力ポートなどである。

【００１７】ハードウェア仕様２において記述されているハードウェアは、１以上のモジュールを含む。モジュールが複数あるときは、それらは、しばしば階層的に記述される。この場合、しばしば、上位階層のモジュールを「親（ペアレント）」、下位階層のモジュールを「子供（チャイルド）」と呼ぶことがある。なお、各モジュールは、それぞれ１以上の資源を含む。

【００１８】プロパティ生成ツール１は、入力されたハードウェア仕様２を解析し、プロパティ（検証ツール３が検証すべき項目または条件式）を抽出する。この実施例では、プロパティは、「資源競合」および「レジスタ漏れ」であり、例えば、ＣＬＴなどの時相論理式（時間経過に伴う信号変化の条件式）で表される。

【００１９】検証ツール３は、既存の論理検証ツールであり、プロパティ生成ツール１により抽出されたプロパティについて検証する。具体的には、ハードウェア仕様２により記述されているハードウェアにおいて「資源競合」または「レジスタ漏れ」が発生するか否かを検証する。なお、検証ツール３は、他の項目についても検証することができる。

【００２０】図２は、プロパティ生成ツール１の動作を説明するフローチャートである。この処理は、ハードウェア仕様２が入力されたときに実行される。ステップＳ１（Phase1）では、ハードウェア仕様２の記述に基づいて、データ転送グラフ（ＤＴＤＡＧ：Data Transfer Directed Acyclic Graph）が作成される。このとき、ハードウェア仕様２において定義されている各モジュール毎に資源が認識され、さらに、モジュールが階層的に記述されているときはそれらの資源の親－子関係も認識される。また、データ転送に係わる条件が考慮される。これらの条件は、ハードウェア仕様２に記述されている。データ転送グラフは、ハードウェア仕様２により記述されているハードウェアに含まれる資源およびデータ転送を表す。

【００２１】ステップＳ２（Phase2）では、ステップＳ１で作成されたデータ転送グラフが最適化（簡略化または最小化）される。具体的には、データ転送グラフにおいて表されている資源およびデータ転送のうち、「資源競合」又は「レジスタ漏れ」が発生する可能性のない資源またはデータ転送が削除される。

【００２２】ステップＳ３（Phase3）では、ステップＳ２において最適化されたデータ転送グラフからプロパティを抽出し、さらにそれらに対応する入力スクリプトを生成する。この入力スクリプトは、検証ツール３が処理可能な形式で記述される。

【００２３】次に、図３を参照しながら、データ転送グラフおよび抽出すべきプロパティについて説明する。図３は、データ転送グラフの一例である。このデータ転送グラフは、入力されたハードウェア仕様に基づいて作成されるが、ここでは、その作成方法の説明は省略する。なお、データ転送グラフは、基本的に、コンピュータ画面に実際に表示されるものではない。したがって、図３

に示すグラフは、データ転送グラフを模式的に示したものである。このことは、以下の図面においても同様である。ただし、データ転送グラフをコンピュータ画面に実際に表示することは可能であり、そのようにしてもよい。

【００２４】各ノードA～F、H、Iは、それぞれデータ資源に対応する。例えば、ノードAは、ハードウェア仕様２において「interface control register file」という名称が与えられたレジスタである。また、ノード間に描かれている矢印は、データ転送に対応する。以下、この矢印を「データ転送エッジ」または単に「エッジ」と呼ぶことにする。また、例えば、ノードAからノードDへのデータ転送に対応するエッジは、ノードAにとってはファンアウトエッジ（または、出力エッジ）であり、同時に、ノードDにとってはファンインエッジ（または、入力エッジ）である。

【００２５】各データ転送エッジには、そのデータ転送についての定義や条件が付与されている。「C」は、データ転送が「同時転送（Concurrent Transfer）」であることを表し、「S」は、そのデータ転送が「順次転送（Sequential Transfer）」であることを表す。また、「Gi」は、データ転送についての動作条件（guardor enabling condition）を表す。たとえば、図３に示す例では、条件G1が満たされたときに、ノードAからノードDへのデータ転送が実行される。

【００２６】上記データ転送グラフにおいて、あるノードに対して複数のファンインエッジが存在する場合（Multiple fan-in）、「資源競合」が発生する可能性がある。ノードDに注目する。ノードDは３２ビットのレジスタ"reg data out"である。ノードDには、エッジADおよびエッジBDが入力されている。したがって、条件G1が満たされたときに、ノードA（レジスタ"interface control registerfile"）に格納されているデータがノードD（レジスタ"reg data out"）に転送され、また、条件G2が満たされたときに、ノードB（入力ポート"data out"）からノードD（レジスタ"reg data out"）へデータが転送される。この場合、もし、任意のクロックサイクルにおいて、条件G1およびG2が同時に満たされると、レジスタ"reg data out"に対して同時に２つの書込アクセスが発生することになる。すなわち、「資源競合」が発生することになる。

【００２７】上述のような「資源競合」は、データ転送グラフでは、複数のファンインエッジが存在するノードにおいて発生し得る。したがって、プロパティ生成ツール１は、データ転送グラフにおいて複数のファンインエッジが存在するノードを検出する。そして、検証ツール３が、そのノードに係わるデータ転送のタイミングを考慮しながら、実際に「資源競合」が発生するか否かを検証する。

【００２８】また、あるノードに対してファンインエッジおよびファンアウトエッジが存在する場合には、「レジスタ漏れ」が発生する可能性がある。例えば、ノードDには、ファンインエッジAD、BD、およびファンアウトエッジDFが存在する。ここで、このハードウェアは、ノードAのデータをノードDを介してノードFに送った後に、ノードBのデータをノードにDに書き込む動作をするものとする。したがって、この動作は、条件G1、G6、G2がその順番で満たされたときに得られる。ところが、もし、条件G6が満たされる前に条件G1および条件G2が満たされてしまうと、ノードDからノードFへのデータ転送は、ノードDがノードBから転送されてくるデータにより上書き（data overwrite）された後に実行されることになる。すなわち、「レジスタ漏れ」が発生することになる。

【００２９】なお、「レジスタ漏れ」は、あるノードに対するファンインエッジが１つの場合にも起こり得る。例えば、第１のノードから第２のノードを介して第３のノードへデータを転送する場合に、第２のノードから第３のノードへのデータ転送が実行される前に、第１のノードから第２のノードへ他のデータが転送されると、上述の場合と同様に、「レジスタ漏れ」が起こり得る。

【００３０】このように、上述のような「レジスタ漏れ」は、データ転送グラフにおいてあるノードに１以上のファンインエッジおよび１以上のファンアウトエッジが存在する場合に発生し得る。したがって、プロパティ生成ツール１は、データ転送グラフからファンインエッジおよびファンアウトエッジが存在するノードを検出する。そして、検証ツール３が、そのノードに係わるデータ転送のタイミングを考慮しながら、実際に「レジスタ漏れ」が発生するか否かを検証する。

【００３１】レジスタに対応するノードに対してファンインエッジおよびファンアウトエッジが存在する場合には、さらに他の問題が内在する。例えば、もし、あるレジスタに対応するノードのファンインエッジに係わる条件と、そのノードのファンアウトエッジに係わる条件とが同一クロックサイクル内で満たされるとすると、そのレジスタに対して書込アクセスおよび読出アクセスが同時に行われることになる。このような状況は、レジスタ転送レベルのモデルでは必ずしもエラーではない。しかし、この状況は、ゲートレベル或いはレイアウトレベルにおいてクロックの歪みを考慮すると、上記レジスタから誤ったデータを読み出してしまう恐れがある。すなわち、もし、クロックが歪んでいると、あるクロックサイクルにおいて、レジスタからのデータ読出しが完全に終わる前にそのレジスタに新たなデータの書込み行われてしまうことがある。この問題は、実際の半導体チップ上では、レジスタ漏れプロパティとして現れる。なお、図３に示す例では、例えば、ノードIからノードCへのデータ転送とノードCからノードDへのデータ転送との間でこの種の「資源競合」が起こり得る。

【００３２】上述のように、「資源競合」および「レジスタ漏れ」が実際に発生するのか否かは、検証ツール３により検証される。すなわち、プロパティ生成ツール１は、これらの問題が発生する可能性のあるプロパティをハードウェア仕様２から自動的に生成し、それらを検証ツールに渡す。

【００３３】なお、上述のプロパティ（「資源競合」および「レジスタ漏れ」）は、記述されたハードウェアの資源およびデータ転送がデータ転送グラフで表された場合、そのグラフのトポロジに関係する。即ち、データ転送グラフのトポロジを解析すれば、「資源競合」または「レジスタ漏れ」が発生する可能性を判断できる。このため、プロパティ生成ツール１は、データ転送グラフのトポロジから自動的にプロパティを生成できる。ただし、この場合、資源およびデータ転送に係わる情報は、レジスタ転送レベルコードから抽出されている必要がある。

【００３４】ところで、ハードウェア仕様２において記述されているハードウェアが大規模になると、必然的に、その記述から生成されるデータ転送グラフは複雑になる。このため、プロパティ生成ツール１は、プロパティを正確に且つ短時間に生成するために、生成したデータ転送グラフを最適化（最小化または簡略化）する。最適化の方法としては、データ転送グラフのトポロジに基づく最適化（トポロジ最適化）、およびデータ転送の動作条件に基づく最適化（論理最適化）が考えられる。

【００３５】図４は、データ転送グラフを最適化する処理のフローチャートである。この処理は、図２のステップＳ２に相当する。ステップＳ１１では、「非フラグメント等価（Unfragmented Equivalence）」を検出する。ステップＳ１２では、「フラグメント等価（Fragmented Equivalence）」を検出する。ステップＳ１３では、「名称等価（Named Equivalence ）」を検出する。ステップＳ１４では「フラグメントエッジ（Fragmented Edges）」を検出する。ステップＳ１５では、「等価エッジ（Edges in Equivalence）」を検出する。ステップＳ１６では、「レジスタファイル（Register File ）」を検出する。上記ステップＳ１１～Ｓ１６により、資源どうしの等価関係が検出される。そして、それらの等価関係を利用し、データ転送グラフから「資源競合」または「レジスタ漏れ」が発生する可能性のない部分を削除する。これにより、データ転送グラフが最適化（すなわち、最小化）される。

【００３６】ここで、ステップＳ１１～Ｓ１６の処理のうちの幾つかについて簡単に説明をしておく。なお、ステップＳ１１～Ｓ１６の各処理は、後述フローチャートを参照しながら詳しく説明する。

【００３７】図５は、ハードウェアの例である。図６は、図５に示すハードウェアを階層的に記述する方法を説明する図である。ハードウェア資源およびデータ転送は、しばしば、図６に示すように、階層的に設計されて記述される。ここでは、「モジュール０」が上位階層であり、「モジュール１」および「モジュール２」が「モジュール０」の下位階層である。

【００３８】このように、階層ごとにデータ資源およびデータ転送が記述される場合、同一の資源またはデータに対して異なる名称が付与されることが多々ある。例えば、図６に示す例においては、モジュール０への入力である"input 1" は、モジュール１においては"input 11" として記述されている。

【００３９】ところで、ハードウェア記述言語では、モジュール間のデータ転送は、通常、入力ポートまたは出力ポートを用いて記述される。たとえば、モジュール０における"input 1" とモジュール１における"input 11" との対応関係は、しばしば、モジュール０の"input port 1"からモジュール１の"input port 11" へのデータ転送として表される。本実施形態では、このようなポート同士の対応関係を「ポート連携（Port Association）」と呼ぶ。そして、この「ポート連携」は、ハードウェア仕様２において定義されている。

【００４０】また、ハードウェア記述言語では、あるモジュール内のレジスタに格納されているデータを他のモジュールへ転送する場合、そのデータは、出力ポートを介して転送されるように記述される。この場合、その出力ポートには、通常、上記レジスタに付与されている名称と同じ名称が付与される。本実施形態では、この対応関係を「名称連携（Named Association ）」と呼ぶ。

【００４１】図７は、「資源の等価」を利用してデータ転送グラフを最適化する例を示す図である。図７(a)は、ハードウェア仕様２において記述されている資源およびデータ転送の定義から生成されたデータ転送グラフを示す。この実施例では、モジュール１には、３つの資源（入力ポートSD-DQ-IN、レジスタdata-in 、出力ポートdata-in ）が設けられている。また、モジュール１の出力は、モジュール２において定義されているバスワイヤdata-buf-outに転送され、さらに、モジュール３において定義されているレジスタreg-data-outに転送される。

【００４２】ここで、レジスタdata-in および出力ポートdata-in は、互いに「名称連携」の関係を有する。すなわち、これら２つの資源は、等価（名称等価）であるものとみなすことができる。したがって、入力ポートSD-DQ-INからレジスタdata-inへのデータ転送は、入力ポートSD-DQ-INから出力ポートdata-in へのデータ転送として扱うことができる。また、出力ポートdata-in およびバスワイヤdata-buf-outは、互いに「ポート連携」の関係を有する。即ち、これら２つの資源は、等価であるものとみなすことができる。従って、入力ポートSD-DQ-INから出力ポートdata-in へのデータ転送は、入力ポートSD-DQ-INからバスワイヤdata-buf-outへのデータ転送

として扱うことができる。そして、これらの等価関係を利用すれば、図７(a) に示すデータ転送グラフは、図７(b) に示す形態に最適化（すなわち、簡略化）される。

【００４３】なお、データ転送の転送元の資源と転送先の資源とが等価であるためには、それらの資源が扱うデータのビット幅が互いに同じである必要がある。従って、以下の定義が得られる。定義１：互いに隣接する１組の資源が互いに同じビット幅仕様を備える場合、それらの等価な資源は、「非フラグメント等価」である。

【００４４】図８は、「フラグメント」を説明する図である。図８に示す例では、中間レベルのモジュールＣに３２ビット幅の出力ポートＯＰが設けられ、その上位レベルのモジュールＢにそれぞれ８ビット幅の出力ポートＯＰ１～ＯＰ４が設けられている。

【００４５】ここで、あるデータ幅を持ったデータ資源とそのデータ幅よりも小さいデータ幅のデータ資源（あるいは、そのデータ幅よりも大きいデータ幅のデータ資源）との間でデータ転送がある場合、「フラグメント等価」が生じ得る。図８に示す例では、モジュールＢの出力ポートＯＰ１～ＯＰ４は、モジュールＣの出力ポートＯＰとの関係において「フラグメント等価」とみなされる。

【００４６】図８に示すモデルにおける出力ポートＯＰ、および出力ポートＯＰ１～ＯＰ４は、データ転送グラフでは、それぞれノードで表され、また、出力ポートＯＰから出力ポートＯＰ１～ＯＰ４へのデータ転送は、それぞれエッジで表される。ここで、上述したように、出力ポートＯＰ１～ＯＰ４にそれぞれ対応するノードが「フラグメント等価」である場合、上記データ転送に対応する４本のエッジはフラグメントエッジである。本実施形態では、「フラグメント等価」であるノードに接続される複数のエッジを、「フラグメントエッジ」と呼ぶ。

【００４７】「フラグメント等価」の有用性は、フラグメントデータ転送を組み合わせることにより非フラグメントデータ転送を得る可能性があることである。また、データ転送の転送元資源または転送先資源が「フラグメント等価」であるか否かに基づいて、データ転送がフラグメントデータ転送であるか非フラグメントデータ転送であるかを識別することができる。

【００４８】フラグメントデータ転送を最適化する他の方法は、「独立フラグメントデータ転送」である。この「独立フラグメントデータ転送」について、以下の定義を設ける。定義２：フラグメントデータ転送エッジの中の任意のエッジの組合せにおいて資源競合を起こさないのであれば、それらのエッジは「独立」である。

【００４９】本実施形態の最適化処理では、データ転送グラフから「独立」なエッジが削除される。これにより、資源競合プロパティまたはレジスタ漏れプロパティが誤って生成されることが回避される。

【００５０】レジスタファイルは、ハードウェア規模を小さくするために、複数のレジスタをグループ化し、バスと各レジスタとの間の接続を簡略化する技術である。これにより、例えば、図９(a) に示す構成が、図９(b) に示す構成に簡略化される。ただし、レジスタファイルに属する各レジスタは、ハードウェア記述言語では、しばしば独立したレジスタとして記述される。この場合、レジスタファイルは、データ転送グラフにおいて、複数のレジスタとして表わされる。このため、データ転送グラフの最適化処理では、レジスタファイルを認識し、そのレジスタファイルに属する複数のレジスタに対応する複数のノードを、１つのノードに置き換える。

【００５１】次に、具体的な実施例を示す。

第１の実施例

図１０～図１５は、プロパティ生成ツール１に入力されるハードウェア仕様２の一例である。ここでは、ハードウェア記述言語として、Verilog が使用されている。

【００５２】図１０は、モジュール０について記述したコードである。このコードのセクション１の記述によれば、モジュール０は、２つの入力ポート（INPUT1,2）、および２つの出力ポート（OUTPUT1,2）を備える。これらの各ポートは、それぞれ１６ビット幅である。また、モジュール０には、クロックおよび制御信号が入力される。さらに、モジュール０には１６ビット幅のバスワイヤCONEKTが設けられている。

【００５３】セクション２の記述によれば、モジュール０の下位階層にモジュール１が設けられている。そして、モジュール０とモジュール１との間におけるポートの対応関係（ポート連携）が定義されている。この例では、例えば、モジュール０の入力ポートINPUT1の８～１５ビットがモジュール１の入力ポートIPORT3に対応付けられ、モジュール０の入力ポートINPUT1の０～７ビットがモジュール１の入力ポートIPORT4に対応付けられている。この場合、入力ポートINPUT1は、入力ポートIPORT3および入力ポートIPORT4の「親資源」であり、また、入力ポートIPORT3および入力ポートIPORT4は、入力ポートINPUT1の「子供資源」である。また、バスワイヤCONEKTがモジュール１の入力ポートIPORT1,2に接続されている。

【００５４】セクション３の記述によれば、モジュール０の下位階層にモジュール２が設けられている。セクション３では、モジュール０とモジュール２との間におけるポートの対応関係が定義されている。また、バスワイヤCONEKTがモジュール２の出力ポートOPORT1に接続されている。

【００５５】図１１および図１２は、モジュール１について記述したコードである。セクション４の記述によれば、各入力ポート（IPORT1-4）、出力ポート（OPORT1,2）はそれぞれ８ビット幅である。また、セクション５の記述によれば、モジュール１には、レジスタＡおよび

レジスタＢが設けられている。そして、レジスタＡに格納されているデータは出力ポートOPORT1に転送され、レジスタＢに格納されているデータは出力ポートOPORT2に転送される。なお、これらのデータ転送は、クロックまたは制御信号に係わらず実行される。すなわち、これらのデータ転送は、同時転送（Concurrent Transfer）タイプに属する。具体的には、"continuousassignment"による転送である。

【００５６】セクション６の記述によれば、制御信号１が与えられると、入力ポートIPORT1に入力されたデータがレジスタＡに書き込まれると共に、入力ポートIPORT3に入力されたデータがレジスタＢに書き込まれる。また、セクション７の記述によれば、制御信号２が与えられると、入力ポートIPORT2に入力されたデータがレジスタＡに書き込まれると共に、入力ポートIPORT4に入力されたデータがレジスタＢに書き込まれる。これらのデータ転送は、制御信号に従って実行される。すなわち、これらのデータ転送は、順次転送（Sequential Transfer）タイプに属す。具体的には、"sequential always block"による転送である。

【００５７】図１３～図１５は、モジュール２について記述したコードである。セクション８の記述によれば、各入力ポート（IPORT5）、出力ポート（OPORT3,4）はそれぞれ１６ビット幅である。また、セクション９の記述によれば、モジュール２は、レジスタＣ～Ｆを備える。そして、レジスタＥに格納されているデータは出力ポートOPORT3に転送され、レジスタＦに格納されているデータは出力ポートOPORT4に転送される。これらのデータ転送は、同時転送である。また、各レジスタＣ～Ｆには、それぞれ所定の初期値が与えられている。

【００５８】セクション１０の記述によれば、制御信号３が与えられると、入力ポートIPORT5に入力されたデータがレジスタＣに書き込まれ、そうでない場合には、入力ポートIPORT5に入力されたデータはレジスタＤに書き込まれる。また、セクション１１の記述によれば、制御信号４が与えられると、レジスタＣ、Ｄに所定の演算結果データが書き込まれる。さらに、セクション１２の記述によれば、レジスタＦにも所定の演算結果データが書き込まれる。

【００５９】セクション１３の記述によれば、制御信号５が与えられると、レジスタＣに格納されているデータがレジスタＥに書き込まれる。また、制御信号５が与えられることなく制御信号６が与えられると、レジスタＤに格納されているデータがレジスタＥに書き込まれる。さらに、制御信号５および６が与えられることなく制御信号７が与えられ場合、および制御信号５～７が与えられなかった場合は、それぞれレジスタＥに所定の定数データが書き込まれる。

【００６０】図１６は、図１０～図１５に示したハードウェア仕様から生成されたデータ転送グラフである。このグラフは、図２に示したフローチャートのステップＳにより生成される。なお、ハードウェア仕様において定義されているデータ資源は、データ転送グラフでは、「ノード」として表される。そして、各ノードごとに、データ資源の名称、データ資源の種別（レジスタ、バスワイヤ、ポートなど）、モジュール名、およびノード名が設定される。また、各データ転送は、データ転送グラフでは、それぞれエッジ（図１６では、「矢印」で描かれている）で表される。

【００６１】なお、このデータ転送グラフは、発明を理解しやすくするために模式的に描いたものであり、実際は、モジュールリスト、データ資源リスト、データ転送エッジリスト、および等価クラスリスト等から構成される。

【００６２】モジュールリストは、ハードウェア仕様において定義されている各モジュールについて以下の情報を格納する。

・モジュールの名称
・親モジュール（当該モジュールの上位階層のモジュール）
・子供モジュール（当該モジュールの下位階層のモジュール）
・階層レベル（当該モジュールが属する階層）
・処理状態（最適化処理における中間状態を表すフラグ等）
・各種ポインタ

例えば、モジュール０についてモジュールリストに格納すべき情報としては、ハードウェア仕様から以下が得られる。

【００６３】
・モジュールの名称：MOD0
・親モジュール：なし
・子供モジュール：MOD1, MOD2
・階層レベル：０

なお、「処理状態」は、最適化処理における中間状態を表すフラグ等であり、ハードウェア仕様から得られるのではない。また、「ポインタ」も、ハードウェア仕様から得られるのではない。

【００６４】データ資源リストは、ハードウェア仕様において定義されている各データ資源について以下の情報を格納する。このリストは、資源の種別（レジスタ、入力ポート、出力ポート、バス、データ転送、定数、．．．）ごとに生成される。

【００６５】
・資源の名称
・親モジュール
・階層レベル
・ビット幅仕様
・親資源
・子供資源

・非フラグメント等価クラス
・フラグメント等価クラス
・エッジ
・処理状態
例えば、入力ポートINPUT1（ノードA）についてデータ資源リストに格納すべき情報としては、ハードウェア仕様から以下が得られる。
【００６６】
・資源の名称：INPUT1
・親モジュール：なし
・階層レベル：０
・ビット幅仕様：「１５：０」
・親資源：なし
・子供資源：IPORT5
・エッジ：エッジＡＦ
なお、「非フラグメント等価クラス」および「フラグメント等価クラス」は、ハードウェア仕様から得られるのではなく、データ転送グラフを解析することにより得られるものである。
【００６７】データ転送エッジリストは、ハードウェア仕様において定義されている各データ転送について以下の情報を格納する。
・エッジの名称
・転送先資源
・転送元資源
・ガード表現（データ転送の条件）
・データ転送タイプ（同時転送または順次転送など）
・転送先または転送元のフラグメント状態
・最適化処理における処理状態
例えば、入力ポートIPORT1（ノードＪ）からレジスタREG-A（ノード０）へのデータ転送に対応するエッジについてデータ転送エッジリストに格納すべき情報としては、ハードウェア仕様から以下が得られる。
【００６８】
・エッジの名称：ＪＯ
・転送先資源：レジスタREG-A（ノード０）
・転送元資源：入力ポートIPORT1（ノードＪ）
・ガード表現：ＣＴＲＬ１
・データ転送タイプ：Ｓ（順次転送）
なお、「転送先または転送元のフラグメント状態」は、データ転送グラフを解析することにより得られる。
【００６９】等価クラスリストは、以下の情報を格納する。なお、これらの情報は、データ転送グラフを解析することにより得られる。
・等価クラス識別子
・各等価クラスに属する資源
・処理状態
データ転送グラフを作成する際には、まず、ハードウェア仕様の各モジュール毎の記述を上位階層から順番に解析してゆき、定義されている各データ資源、それら資源の対応関係、およびそれらの資源を利用したデータ転送を認識する。このとき、あるモジュールから他のモジュールへのコールがあるか否かを調べるために、behavioural 記述がスキャンされる。そして、そのようなコールがあった場合には、呼び出されたモジュール毎に、サーチキューを設け、先に認識されている資源との間でインタフェース定義の対応関係を認識する。
【００７０】また、各ポート（入力ポート、出力ポート、入出力ポートを含む）、バスワイヤ、レジスタのビット幅を認識し、さらに、各バスワイヤごとに、そのバスワイヤに対応するドライバおよびレジスタを認識する。
【００７１】ハードウェア資源２から抽出した情報は、モジュールリスト、データ資源リスト、データ転送エッジリストに格納される。図１０～図１５に示す例では、これらの処理により、図１６に示す各ノードＡ～Ｙ（Ｅ、Ｇ、Ｐ、Ｑ、Ｒを除く）が生成され、それぞれビット幅が設定される。
【００７２】さらに、データ転送毎に設定されている条件を解析する。これらの条件は、たとえば、"gurded continuous assignment statement"、"combinational alwausblock"、"sequential always block" である。これらの情報は、「ガード表現」として、データ転送毎にデータ転送エッジリストに登録される。これにより、各データ転送に対応するエッジが生成されることになる。なお、条件が設定されていないデータ転送については、そのデータ転送に対して「１」が付与される。
【００７３】ハードウェア仕様において代入式が記述されている場合（例えば、セクション１０およびセクション１１）には、図１７(a) に示すように、疑似資源Ｓdtが生成される。この疑似資源Ｓdtは、記述されている代入式を実装する仮想的なデータソースである。たとえば、セクション１１に記述されている代入式に対応するノードＰ（Sdt3）が生成される。ノードＥ、Ｇも同様である。また、あるレジスタに予め決められた定数が書き込まれる場合（例えば、セクション１３）は、図１７(b) に示すように、疑似資源Ｓdcが生成される。この疑似資源Ｓdcもまた仮想的なデータソースである。例えば、セクション１３に記述された定義に対応してノードＱ、Ｒが生成される。なお、これらの疑似資源は、ハードウェア仕様に基づいてデータ資源リストに所定の情報を登録することにより生成される。
【００７４】続いて、上述のようにして生成されたデータ転送グラフが最適化される。図１６に示す例では、データ転送グラフにおいて「非フラグメント等価」「フラグメント等価」および「独立エッジ」を検出することによりデータ転送グラフが最適化される。
【００７５】「非フラグメント等価」を検出する場合、ハードウェア仕様のインタフェース定義において互いに「等価」であると定義されている１組の資源であって、

データ幅が互いに同じであるものを認識する。この場合、対象となる資源は、たとえば、入力ポート、出力ポートおよび入出力ポートである。そして、非フラグメント等価クラスに属する資源が検出されると、そのことを表す情報が、データ資源リストにおいて、その資源に対応する記憶領域に登録される。

【００７６】上記等価関係は、各ポート毎に調べられる。このとき、各ポート毎に処理は、最終資源にたどり着くまで、あるいは異なるビット幅の資源に到達するまで継続される。

【００７７】ここで、図１６に示したグラフにおける「非フラグメント等価」を説明する。ノードＡ（モジュール０のINPUT2）及びノードＦ（モジュール２のIPORT 5）は、共に１６ビットの入力ポートである。また、これらの入力ポートは、ハードウェア仕様のセクション３に定義されているように、互いに対応する資源である。即ち、ノードＦに対応する資源は、ノードＡに対応する資源の「子供」である。従って、ノードＡおよびノードＦは、「非フラグメント等価」である。同様に、ノードＣはノードＤと等価であり、ノードＸはノードＷと等価である。

【００７８】この場合、例えば、資源Ａおよび資源Ｆが非フラグメント等価クラス＃１に属しているとすると、データ資源リストにおいて、資源Ａおよび資源Ｆの「非フラグメント等価クラス」にそれぞれ「＃１」が登録される。

【００７９】「フラグメント等価」を検出する場合は、分岐されている資源の中から等価なものを認識する。図１６に示す例では、例えば、ノードＢからノードＨおよびノードＩへデータが転送されている。このとき、ノードＢのビット仕様が１６ビット幅であるのに対し、ノードＨおよびノードＩのビット仕様はそれぞれ８ビット幅である。即ち、各転送先のノードで使用されるビット幅は、転送元のノードで使用されるビット幅よりも小さい。従って、ノードＨおよびノードＩは、「フラグメント等価」である。このことは、ノードＪ、Ｋにおいても同様である。

【００８０】一方、ノードＹには、ノードＳおよびノードＴからデータが転送されてきている。このとき、各転送元のノードで使用されるビット幅は、転送先のノードで使用されるビット幅よりも小さい。従って、ノードＳおよびノードＴは、「フラグメント等価」である。

【００８１】この場合、例えば、資源Ｈおよび資源Ｉがフラグメント等価クラス＃５に属しているとすると、データ資源リストにおいて、資源Ａおよび資源Ｆの「フラグメント等価クラス」にそれぞれ「＃５」が登録される。

【００８２】「独立エッジ」は、以下のようなものである。たとえば、ノードＰはノードＵのみにデータを転送し、ノードＵはノードＰのみからデータを受け取る。この場合、ノードＰからノードＵへのデータ転送に対応するエッジは、独立エッジである。また、ノードＹには、ノードＳ、Ｔからデータが転送されてきている。このとき、ノードＳからは０～７ビットが転送され、ノードＴには８～１５ビットが転送されてきている。すなわち、これら２つのデータ転送においてデータビットはオーバラップしていない。この場合、ノードＳからノードＹへのデータ転送に対応するエッジ及びノードＴからノードＹへのデータ転送に対応するエッジは、共に独立エッジである。図１６に示すグラフでは、この他にも、例えば、ノードＮからノードＳへのデータ転送、ノードＯからノードＴへのデータ転送、ノードＵからノードＤへのデータ転送、ノードＶからノードＷへのデータ転送に対応するエッジもそれぞれ独立エッジである。

【００８３】「独立エッジ」と関連して、「フラグメントエッジ」および「等価エッジ」の概念を導入する。「フラグメントエッジ」とは、転送元の資源または転送先の資源がフラグメント等価クラスに属するエッジのことを指す。例えば、図１６に示す例では、ノードＪおよびノードＫがフラグメント等価クラスに属しているので、この場合、エッジＣＪ、エッジＣＫ、エッジＪＯ、エッジＫＯは、この等価クラスに係わるフラグメントエッジである。なお、各エッジは、(1) 転送元および転送先の資源が共に「フラグメント等価」でない状態、(2) 転送元の資源のみが「フラグメント等価」である状態、(3) 転送先の資源のみが「フラグメント等価」である状態、(4) 転送元および転送先の資源が共に「フラグメント等価」である状態に分類される。なお、この分類は、エッジ毎に２ビットのフラグで表される。

【００８４】「等価エッジ」とは、転送元の資源または転送先の資源がフラグメント等価クラスに属するノードであるエッジの集合である。例えば、ノードＪ、Ｋはフラグメント等価クラスに属するので、この場合、エッジＣＪ、エッジＣＫ、エッジＪＯ、エッジＫＯが、この等価クラスに係わる等価エッジである。なお、独立エッジは、等価エッジに属するエッジの集合から除去される。

【００８５】上述のようにして等価関係を検出した後、データ転送グラフから「資源競合」または「レジスタ漏れ」に関わりのない部分を除去する。具体的には、ノードＡおよびノードＦは「非フラグメント等価」である。したがって、ノードＡが除去される。また、エッジＶＷが「独立エッジ」であり、ノードＷおよびノードＸは「非フラグメント等価」である。したがって、ノードＷ、Ｘが除去される。さらに、エッジＰＵおよびエッジＵＤがそれぞれ「独立エッジ」であり、ノードＤおよびノードＣは「非フラグメント等価」である。したがって、ノードＰ、Ｕ、Ｄが除去される。

【００８６】また、ノードＨ、Ｉは、ノードＢから見て「フラグメント等価」である。従って、ノードＢが除去される。同様に、ノードＣも除去される。また、ノード

Ｓ、Ｔは、ノードＹから見て「フラグメント等価」である。従って、ノードＹが除去される。さらに、エッジＮＳおよびエッジＯＴは共に独立エッジである。したがって、ノードＳ、Ｔが除去される。

【００８７】上述の処理の結果、図１６に示すデータ転送グラフは、図１８に示す状態に最適化される。上述の処理は、具体的には、データ資源リストおよびデータ転送エッジリスト等からエッジリストＥfinal を生成する処理に相当する。エッジリストＥfinal は、最適化処理の結果として得られるエッジのリストである。

【００８８】最適化処理が終わると、図２のフローチャートのステップＳ３の処理が実行される。この処理は、図１９に示すように、２つのステップからなる。ステップＳ２１では、最適化されたデータ転送グラフから隣接ノードリストを作成する。続いて、ステップＳ２２において、プロパティおよびそれに対応する検証スクリプトを生成する。本実施形態では、「資源競合」または「レジスタ漏れ」が発生し得るすべての条件を抽出する。具体的には、隣接ノードリストを利用して各ノード毎の入力エッジ数および出力エッジ数をカウントし、それに基づいてプロパティを生成する。そして、抽出した「資源競合」または「レジスタ漏れ」が発生し得る条件が、検証ツール３が実行可能なスクリプトに変換される。

【００８９】図１８に示すグラフからプロパティを生成する場合の処理を説明する。まず、最適化されたグラフから隣接ノードリストが作成される。隣接ノードリストは、データ転送グラフ上の各エッジ毎に、その転送元ノードおよび転送先ノードを検出することにより得られる。図２０に隣接ノードリストの例を示す。なお、各エッジ毎の転送先ノードおよび転送元ノードに係わる情報は、最適化処理により得られるエッジリストＥfinal に格納されている。

【００９０】続いて、隣接ノードリストを利用し、「資源競合」または「レジスタ漏れ」が発生する可能性があるノードを検出する。「資源競合」を検出する場合には、複数の入力エッジが存在するノードを検出する。図２０に示す例では、ノードＬ、Ｍ、Ｎ、Ｏにおいてそれぞれ２本に入力エッジが存在し、ノードＶにおいて４本のエッジが存在する。従って、これら５つのノードは、それぞれ「資源競合」が発生する可能性があるものとみなされる。

【００９１】一方、「レジスタ漏れ」を検出する場合には、１以上の入力エッジが存在し、且つ１以上の出力エッジが存在するノードを検出する。図２０に示す例では、ノードＬ、Ｍにおいて、２本の入力エッジおよび１本の出力エッジが存在する。したがって、これらのノードは、「レジスタ漏れ」が発生する可能性があるものとみなされる。なお、クロックの歪みを考慮すると、「レジスタ漏れ」の可能性があるノード（レジスタ）は、「資源競合」が発生する可能性があるものとみなされる。

【００９２】図２１は、「資源競合」のプロパティの例である。これらのプロパティは、図１８に示すデータ転送グラフから生成されたものである。「資源競合」のプロパティは、上述した５つのノードについてそれぞれ生成される。このとき、各ノードにおいて存在する入力エッジに付与されている条件が抽出され、それらの条件を組み合わせることによりプロパティが生成される。

【００９３】例えば、ノードＬについてのプロパティを生成する場合、エッジＥＬおよびエッジＦＬに付与されている条件がグラフから抽出される。すなわち、エッジＥＬおよびエッジＦＬからそれぞれ「制御信号CTRL4 」および「制御信号CTRL3 」が抽出され、それらを組み合わせることにより下記のプロパティが得られる。

【００９４】ＡＧ（ＣＴＲＬ４．ＣＴＲＬ３）

検証ツール３は、このプロパティが入力されると、制御信号CTRL4 および制御信号CTRL3 が同時に発生するクロックサイクルが存在するか否か検証する。そして、それらが同時に発生するクロックサイクルが存在する場合は、検証ツール３は、ノードＬにおいて「資源競合」が発生するものと判断する。即ち、レジスタＣに対して同時書込アクセスが発生するものとみなす。

【００９５】なお、あるノードにおいて３本以上の入力エッジが存在する場合には、それらの中から任意の２本の入力エッジを抽出し、その２本の入力エッジに付与されている条件がグラフから抽出される。この処理は、すべての組合せについて行われる。例えば、ノードＶは４本の入力エッジが存在するので、この場合、６通りの組合（エッジＱＶとエッジＬＶ、ＱＶとＭＶ、ＱＶとＲＶ、ＬＶとＭＶ、ＬＶとＲＶ、ＭＶとＲＶ）が得られる。そして、各組合せごとにプロパティが生成される。

【００９６】図２２は、「レジスタ漏れ」のプロパティの例である。これらのプロパティもまた、図１８に示すデータ転送グラフから生成されたものである。なお、この実施例で検出される「レジスタ漏れ」は、「書込み、読出し、書込み」であるはずのシーケンスが「書込み、書込み、読出し」というシーケンスで実行されてしまう場合を想定する。

【００９７】ノードＬに着目した場合、「レジスタ漏れ」が発生する可能性がある条件は、以下の４つである。すなわち、(1) ノードＥからノードＬへデータが転送された後に、ノードＬからノードＶへの転送が実行されることなく、再びノードＥからノードＬへデータが転送される場合、(2) ノードＦからノードＬへデータが転送された後に、ノードＬからノードＶへの転送が実行されることなく、再びノードＦからノードＬへデータが転送される場合、(3) ノードＦからノードＬへデータが転送された後に、ノードＬからノードＶへの転送が実行されることなく、ノードＥからノードＬへデータが転送される場合、(4) ノードＥからノードＬへデータが転送さ

れた後に、ノードＬからノードＶへの転送が実行されることなく、ノードＦからノードＬへデータが転送される場合である。なお、ノードＭにおいても、基本的に同様にプロパティが生成される。例えば、(1) の場合のプロパティは、

ＥＦ（ＣＴＲＬ４／＼ＥＸ（Ｅ（～ＣＴＲＬ５　Ｕ　ＣＴＲＬ４））

と表される。

【００９８】検証ツール３は、このプロパティが入力されると、「制御信号CTRL4 が発生するクロックサイクルの後に制御信号CTRL5 が発生することなく制御信号CTRL4 が発生するクロックサイクル」を探す。そして、そのような状況が存在するのであれば、検証ツール３は、ノードＬにおいて「レジスタ漏れ」が発生するものと判断する。

【００９９】図２３は、クロックの歪みを考慮した場合におけるレジスタにおける「資源競合」のプロパティの例である。これらのプロパティもまた、図１８に示すデータ転送グラフから生成されたものである。なお、この場合の「資源競合」は、あるクロックサイクルにおいて、書込アクセスおよび読出アクセスが発生する場合を想定する。

【０１００】ノードＬに着目した場合、「資源競合」が発生する可能性がある条件は、以下の２つである。すなわち、(1) ノードＥからノードＬへデータ転送と、ノードＬからノードＶへのデータ転送が動イルクロックサイクル内に実行される場合、および(2) ノードＦからノードＬへデータ転送と、ノードＬからノードＶへのデータ転送が動イルクロックサイクル内に実行される場合である。なお、ノードＭにおいても基本的に同様にプロパティが生成される。

【０１０１】例えば、(1) の場合のプロパティは、

ＡＧ（ＣＴＲＬ４．ＣＴＲＬ５）

として表される。検証ツール３は、このプロパティが入力されると、制御信号CTRL4 および制御信号CTRL5 が同時に発生するクロックサイクルを探す。そして、そのようなクロックサイクルが存在するのであれば、検証ツール３は、ノードＬにおいて「資源競合」が発生するものと判断する。

<u>第２の実施例</u>

図２４～図２６は、プロパティ生成ツール１に入力されるハードウェア仕様２の一例である。この実施例でも、ハードウェア記述言語として、Verilog が使用されている。

【０１０２】セクション１においては、入力される制御信号、入力データおよび出力データが定義されている。また、セクション１の記述によれば、このモジュールには５つのレジスタ（BASES, ENDS, MODE, REG-ADR, REGA-OUT）が設けられている。セクション２には、制御信号CNT-REG-ACC および制御信号WRITE が与えられたときのデータ転送が記述されている。これらのデータ転送は、５ビットの制御信号REG-ADR の値により制御される。また、セクション３には、各種制御信号が与えられたときのレジスタREG-ADR または出力REGA-OUTへのデータ転送が記述されている。

【０１０３】図２７は、図２４～図２６に示したハードウェア仕様から生成されたデータ転送グラフである。ハードウェア仕様からデータ転送グラフを生成する方法は、第１の実施例の場合と同じなので、ここではその説明を省略する。

【０１０４】図２７に示すグラフにおいて、「Ｇ１」～「Ｇ１１」は、データ転送のトリガとしての条件である。これらの条件を図２８に示す。例えば、ノードＡからノードＤへのデータ転送は、セクション２の記述によれば、制御信号CNT-REG-ACC および制御信号WRITE が与えられ、且つレジスタREG-ADR の第４ビットが"1" であり、且つレジスタREG-ADR の第３～０ビットが"0001"であったクロックサイクルにおいて実行される。図２８に示す「Ｇ１」は、この条件を記述している。

【０１０５】なお、各条件に含まれる要素の集合を「サポートセット」と呼ぶ。たとえば、条件Ｇ１のサポートセットは、７つの要素（CNT-REG-ACC, WRITE, REG-ADR(4),REG-ADR(3), REG-ADR(2), REG-ADR(1), REG-ADR(0)）を含んでいる。

【０１０６】このデータ転送グラフを最適化する際には、「レジスタファイル」および「名称等価」の概念が利用される。「レジスタファイル」を検出する場合、まず、転送先がレジスタであるエッジまたは転送元がレジスタであるエッジを抽出する。そして、それらのエッジに付与されているサポートセットを比較し、それらが互いに同じであるエッジ同士をグループ化する。この場合、グループ化された複数のエッジにそれぞれ接続されるレジスタを「レジスタファイル」とみなす。

【０１０７】図２９を参照しながら具体例を示す。ここでは、転送先ノードがレジスタであるエッジＡＤに注目する。エッジＡＤには、条件Ｇ１が付与されている。この場合、レジスタファイルを検出する場合には、まず、レジスタを表すノードへのファンインエッジの中から、Ｇ１のサポートセットと同じサポートセットを有するエッジを抽出する。条件Ｇ１のサポートセットは、図２８に示すように、条件Ｇ２～Ｇ６とそれぞれ同じである。従って、レジスタを表すノードへのファンインエッジエッジのうち、条件Ｇ２～Ｇ６が設定されているエッジが抽出される。これにより、条件Ｇ１～Ｇ６が設定されているエッジがグループ化される。すなわち、エッジＡＤ、ＡＥ、ＡＦがグループ化される。したがって、これらのエッジの転送先である３つのノード（Ｄ、Ｅ、Ｆ）は、レジスタファイルに属するものとみなされる。

【０１０８】なお、上記の例では、レジスタへのファンインエッジに基づいてレジスタファイルを検出したが、

図２９に示すように、レジスタからのファンアウトエッジを利用しても同様の関係が得られる。

【０１０９】「名称等価」を検出する場合は、同一モジュール内のノードの中で同一の名称が付与されているノード（特に、レジスタおよび出力ポート）を抽出し、それらを等価な資源とみなす。図２７に示す例では、レジスタに対応するノードＪおよび出力ポートに対応するノードＩの名称が共に"REGA-OUT なので、これらのノードが等価であるとみなされる。

【０１１０】上述のようにして等価関係を検出した後、データ転送グラフから「資源競合」または「レジスタ漏れ」に関わりのない部分を除去する。すなわち、ノードＤ、Ｅ、Ｆは、「レジスタファイル」なので、これらのノードは１つのノード（ノードＤ'）に置きかえられる。また、ノードＪおよびノードＩは「名称等価」である。したがって、ノードＩが除去される。上述の最適化処理の結果、図２７に示すデータ転送グラフは、図３０に示す状態に最適化される。

【０１１１】データ転送グラフが最適化されると、それに伴って各エッジに付与されている条件（サポートセット）も変化する。具体的には、図２７に示されていた条件Ｇ１～Ｇ６が条件Ｇ１'に集約され、一方、条件Ｇ７およびＧ８が条件Ｇ２'に集約される。図３１に条件Ｇ１'および条件Ｇ２'のサポートセットを示す。

【０１１２】第２の実施例のハードウェアでは、ノードＪにおいて「資源競合」が発生する可能性がある。この場合、「資源競合」のプロパティは、４つの条件（Ｇ２'、Ｇ９、Ｇ１０、Ｇ８）の中の任意の２つが同時に発生するクロックサイクルを調べるためのスクリプトである。また、このハードウェアでは、ノードＤ'において「レジスタ漏れ」が発生する可能性がある。この場合、「レジスタ漏れ」のプロパティは、条件Ｇ１'および条件Ｇ２'が所定の順番で発生することを調べるためのスクリプトである。

【０１１３】次に、上述したプロパティ生成ツール１の動作アルゴリズムをフローチャートを参照しながら説明する。なお、以下の説明では、下記の定義を用いる。

Ｌ　モジュールの階層レベル
Ｍ　モジュールのセット
Ｓｉ　入力ポートのセット
Ｓｏ　出力ポートのセット
Ｓｉｏ　入出力ポートのセット
Ｓｗ　バスワイヤのセット
Ｓｒ　レジスタのセット
Ｓｄｔ　データ要素（代入式等の演算の結果を生成する資源）
Ｓｄｃ　データ定数（定数を生成する資源）
Ｅｄｔ　データ転送エッジ
Ｎ　ノードのセット
Ｈ　データ資源の等価クラスのセット

図３２～図３４は、データ転送グラフを生成する方法を説明するフローチャートである。この処理は、ハードウェア記述言語で記述されたハードウェア仕様２が入力されたときに実行される。

【０１１４】ステップＳ３１では、プロパティを作成するために利用するメモリ領域を初期化する。このとき、モジュール階層Ｌとして「０」が設定される。また、最小ビット幅パラメータとして「Ｗｂ」が設定される。

【０１１５】ステップＳ３２では、モジュールリストＭにルートモジュールを登録する。モジュールリストＭには、ハードウェア仕様２に設けられているすべてのモジュールが登録されている。ステップＳ３３は、モジュールリストＭに登録されている各モジュールについてステップＳ３４以降の処理を実行するための判断処理である。

【０１１６】ステップＳ３４では、モジュールリストＭからモジュールＭｉ を取り出す。ステップＳ３５は、モジュールＭｉ についてのインタフェース定義に記述されている各ポートについてステップＳ３６～Ｓ４１を実行するための処理である。ステップＳ３６では、インタフェース定義からポートＰを抽出する。そして、そのポートＰのビット幅が最小ビット幅パラメータＷｂ 以上か否か調べる。ポートＰのビット幅が最小ビット幅パラメータＷｂ 以上であればステップＳ３７へ進み、そうでない場合は、ステップＳ３５へ戻って次のポートを抽出する。

【０１１７】ステップＳ３７では、ポートＰが入力ポートであるか否かを調べる。ポートＰが入力ポートであった場合には、ステップＳ４０において、ポートＰを入力ポートリストＳｉ に登録する。また、ステップＳ３８では、ポートＰが出力ポートであるか否かを調べる。ポートＰが出力ポートであった場合には、ステップＳ４１において、ポートＰを出力ポートリストＳｏ に登録する。ポートＰが入力ポートまたは出力ポートのいずれでもなかった場合には、ステップＳ３９において、ポートＰを入出力ポートリストＳｉｏに登録する。

【０１１８】上記処理により、ハードウェア仕様２において定義されている各ポートが、入力ポートリストＳｉ、出力ポートリストＳｏ、または入出力ポートリストＳｉｏに登録される。

【０１１９】ステップＳ５１は、モジュールＭｉ のモジュール宣言に記述されている各変数Ｖについてステップ Ｓ５２～Ｓ５５を実行するための処理である。ステップＳ５２では、変数Ｖのビット幅が最小ビット幅パラメータＷｂ 以上であるか否かを調べる。変数Ｖのビット幅が最小ビット幅パラメータＷｂ 以上であればステップＳ５３へ進み、そうでない場合はステップＳ５１に戻って次の変数を抽出する。

【０１２０】ステップＳ５３では、変数Ｖがレジスタ変数であるか否かを調べる。変数Ｖがレジスタ変数であれ

ば、ステップＳ５５においてその変数Ｖをレジスタ変数リストＳｒ に登録し、そうでない場合は、ステップＳ５４においてその変数Ｖをバスワイヤ変数リストＳｗ に登録する。

【０１２１】上記処理により、ハードウェア仕様２において定義されている各変数が、レジスタ変数リストＳｒ またはバスワイヤ変数リストＳｗ に登録される。ステップＳ５６は、モジュールMi の各モジュールインスタンシエイションについてステップＳ５７およびＳ５８を実行するためにの処理である。ステップＳ５７では、モジュールインスタンシエイションm-j をモジュールリストMに登録する。ステップＳ５８では、モジュールインスタンシエイションm-j のポートをモジュールMi 内の先に識別されている資源に関連づける。

【０１２２】ステップＳ６１は、モジュールMi 内の各同時処理ＣＰについてステップＳ６２～Ｓ７２を実行するための処理である。ステップＳ６２では、同時処理ＣＰがcontinuous assignment タイプであるか否かを調べる。ここで、同時処理ＣＰがcontinuous assignment タイプであれば、ステップＳ７１において、その処理ＣＰのデータ転送処理タイプとして「Ｃ（Concurrent）」を設定する。そして、ステップＳ７２において、データ転送エッジを生成する。

【０１２３】ステップＳ６３では、同時処理ＣＰがguarded continuous assignment タイプであるか否かを調べる。ステップＳ６４では、同時処理ＣＰが、combinationalalways blockタイプであるか否かを調べる。そして、ステップＳ６３またはＳ６４の判断結果が「Ｙｅｓ」ならば、ステップＳ７０において、その処理ＣＰのデータ転送処理タイプとして「Ｃ」を設定する。一方、ステップＳ６３およびＳ６４の判断結果が共に「Ｎｏ」ならば、ステップＳ６５において、その処理ＣＰのデータ転送処理タイプとして「Ｓ（Sequential）」を設定する。

【０１２４】ステップＳ６６では、すべてのガード条件を認識する。ここで、各ガード条件は、それぞれ異なるデータ転送を制御する。ステップＳ６７では、各ガード条件毎に、ガード表現およびガード信号を格納する。このとき、このガード条件により制御されるデータ転送の転送元および転送先を認識する。そして、ステップＳ６８において、データ転送エッジを生成する。ステップＳ６９は、各データ転送についてステップＳ６６～Ｓ６８を実行するための処理である。

【０１２５】上記処理により、ハードウェア仕様２において定義されている各データ転送に対応するデータ転送エッジが生成される。図３５は、ステップＳ６８またはＳ７２のデータ転送エッジを生成する処理の詳細フローチャートである。

【０１２６】ステップＳ８１では、データ転送の転送元が代入式であるか否かを調べる。転送元が代入式である場合は、ステップＳ８２において、その代入式に対応する擬似資源Ｓdtを生成する。ステップＳ８３では、データ転送の転送元が定数であるか否かを調べる。転送元が定数である場合は、ステップＳ８４において、その定数に対応する擬似資源Ｓdcを生成する。

【０１２７】データ転送の転送元が、代入式または定数のいずれでもない場合は、ステップＳ８５において、転送元資源Sourceを認識する。ステップＳ８６では、データ転送の転送元が、入力ポートリストＳｉ 、入出力ポートリストＳio、バスワイヤ変数リストＳｗ またはレジスタ変数リストＳｒ に属しているか否かを調べる。

【０１２８】擬似資源Ｓdtまたは擬似資源Ｓdcが生成された場合、あるいはステップＳ８６の判断結果が「Ｙｅｓ」であった場合には、ステップＳ８７において、転送先資源Sinkを認識する。ステップＳ８８では、データ転送の転送先が、出力ポートリストＳｏ 、入出力ポートリストＳio、バスワイヤ変数リストＳｗ またはレジスタ変数リストＳｒ に属しているか否かを調べる。

【０１２９】ステップＳ８８の判断結果が「Ｙｅｓ」であった場合には、ステップＳ８９において、データ転送の転送元および転送先に基づいてエッジＥを生成する。ステップＳ９０では、データ転送エッジリストＥdtに生成したエッジＥを追加する。ステップＳ９１では、エッジＥをデータ転送の転送元及び転送先に関連づける。なお、この実施例では、データ転送の転送元または転送先がポート、レジスタ、バスワイヤ、または疑似資源のいずれでもなかった場合は、エッジＥは生成されない。

【０１３０】上記処理により、ハードウェア仕様２において定義されている各データ転送に対して、そのデータ転送の転送元および転送先が定義されたデータ転送エッジが生成される。

【０１３１】図３６および図３７は、「非フラグメント等価」を検出する処理のフローチャートである。このフローチャートは、図４に示したフローチャートのステップＳ１１の実施例である。なお、ここでは、入出力ポートについて説明するが、入力ポートおよび出力ポートについても同様の処理が実行される。

【０１３２】ステップＳ１０１では、入出力ポートリストＳioをスキャンする。ステップＳ１０２は、入出力ポートリストＳioに属する各要素（入出力ポート）についてステップＳ１０３以降の処理を実行するための処理である。ステップＳ１０３では入出力ポートリストＳioから要素Ｓio-kを抽出する。

【０１３３】ステップＳ１０４では、抽出した要素Ｓio-kを待ち行列Ｑに加える。ステップＳ１０５では、待ち行列Ｑが空か否かを調べる。待ち行列Ｑが空のときはステップＳ１０２に戻り、待ち行列Ｑに１以上の要素が保持されている場合は、ステップＳ１０６において、待ち行列Ｑから要素ｑを抽出する。

【０１３４】ステップＳ１０７では、要素ｑが「子供資

源」を有するか否かを調べる。ここで、要素ｑの「子供資源」とは、例えば、要素ｑが設けられているモジュールの階層の下位の階層のモジュール内に設けられている資源であって、要素ｑに対応付けられているものをいう。対応関係は、ハードウェア仕様などに記述されている。

【０１３５】要素ｑが「子供資源」を有していなければ、ステップＳ１１４において待ち行列Ｑから要素ｑを削除した後にステップＳ１０２に戻り、要素ｑが「子供資源」を有している場合にはステップＳ１０８へ進む。ステップＳ１０８では、要素Ｓio-kに対応する非フラグメント等価クラスHekを生成する。そして、ステップＳ１０９において、Eqvl フラグをリセットする。

【０１３６】ステップＳ１１０は、要素ｑが有する各「子供資源」についてステップＳ１２１～Ｓ１２６を実行するための処理である。ステップＳ１２１では、要素ｑから「子供資源qc」を抽出する。ステップＳ１２２では、要素ｑのビット幅と「子供資源qc」のビット幅が互いに同じであるか否かを調べる。これらのビット幅が互いに異なる場合は、ステップＳ１１０に戻って次の「子供資源」を抽出する。上記ビット幅が互いに同じ場合、ステップＳ１２３においてEqvl フラグに「１」を設定する。ステップＳ１２４では、「子供資源qc」を待ち行列Ｑの要素として追加する。ステップＳ１２５では、「子供資源qc」を非フラグメント等価クラスHekの要素として追加する。ステップＳ１２６では、非フラグメント等価クラスHekと「子供資源qc」とを関連づける。

【０１３７】全ての「子供資源」についてステップＳ１２１～Ｓ１２６の処理が実行されると、ステップＳ１１１において、Eqvl フラグが「１」か否かを調べる。Eqvlフラグが「１」であれば、ステップＳ１１２において要素ｑを非フラグメント等価クラスHekに追加し、さらに、ステップＳ１１３において非フラグメント等価クラスHekと要素ｑとを関連づける。なお、Eqvl フラグが「１」でなかった場合には、ステップＳ１１２およびＳ１１３の処理はスキップされる。

【０１３８】このように、あるポートのビット幅とそのポートの「子供資源」のビット幅が互いに同じである場合、そのポートと「子供資源」は、「非フラグメント等価」とみなされる。

【０１３９】図３８～図４１は、「フラグメント等価」を検出する処理のフローチャートである。このフローチャートは、図４に示したフローチャートのステップＳ１２の実施例である。なお、ここでは、入出力ポートについて説明するが、入力ポートおよび出力ポートについても同様の処理が実行される。

【０１４０】ステップＳ１３１では、入出力ポートリストSioをスキャンする。ステップＳ１３２は、入出力ポートリストSioに属する各要素（入出力ポート）についてステップＳ１３３以降を実行するための処理である。ステップＳ１３３では、入出力ポートリストSioから要素Ｓio-lを抽出する。

【０１４１】ステップＳ１３４では、要素Ｓio-lがいずれかの等価クラスHekに属しているか否かを調べる。要素Ｓio-lがいずれかの等価クラスHekに属している場合は、ステップＳ１３５に進み、そうでない場合には、ステップＳ１４１～Ｓ１４６の処理を実行した後にステップＳ１３２に戻る。

【０１４２】ステップＳ１４１では、要素Ｓio-lが「子供資源」を有するか否かを調べる。要素Ｓio-lが「子供資源」を有する場合は、ステップＳ１４２以降の処理を実行し、そうでない場合は、ステップＳ１３２に戻る。ステップＳ１４２では、要素Ｓio-lに対応するフラグメント等価クラスＯＩＦHelを生成する。ステップＳ１４３では、要素Ｓio-lをルート（出発点）として、要素Ｓio-lのビット幅よりも小さいビット幅を持つ「子供資源」を認識する。ステップＳ１４４では、各「子供資源」のためのフラグメント等価フィールドに「フラグメント等価クラスＯＩＦHel」を設定する。ステップＳ１４５では、各「子供資源」のための非フラグメント等価フィールドに「非フラグメント等価クラスHek」を設定する。ステップＳ１４６では、各「子供資源」をフラグメント等価クラスＯＩＦHelの要素として追加する。

【０１４３】要素Ｓio-lがいずれかの等価クラスHekに属している場合は、ステップＳ１３５において、フラグメントchild-found フラグをリセットする。ステップＳ１３６では、要素Ｓio-lをルートとして、要素Ｓio-lのビット幅よりも小さいビット幅を持った「子供資源」または要素Ｓio-lのビット幅よりも大きいビット幅を持った「子供資源」を探す。このとき、抽出された「フラグメント子供」の集合を"F-child" とする。一方、より大きなビット幅を持った「非フラグメント子供資源」の集合を"UG-child"とする。

【０１４４】ステップＳ１３７では、集合F-child の要素が存在するか否かを調べる。集合F-child の要素が存在するのであれば、ステップＳ１３８においてフラグメントchild-found フラグをセットする。ステップＳ１３９では、要素Ｓio-lに対応するフラグメント等価クラスＯＩＦHelを生成する。

【０１４５】ステップＳ１５１では、集合F-child をスキャンする。ステップＳ１５２は、集合F-child に属する各要素についてステップＳ１５３～Ｓ１５６を実行するための処理である。ステップＳ１５３では、集合F-child から要素Cd-i を抽出する。ステップＳ１５４では、要素Cd-i の「親資源Ｐ－Cd-i 」を抽出する。ステップＳ１５５では、要素Cd-i および親資源Ｐ－Cd-i のためのフラグメント等価フィールドに「フラグメント等価クラスＯＩＦHel」を設定する。ステップＳ１５６では、要素Cd-i をフラグメント等価クラスＯＩＦHelに加える。

【０１４６】集合F-child の要素が存在しない場合は、ステップＳ１６１～Ｓ１７２の処理が実行される。ステップＳ１６１～Ｓ１６５の処理は、基本的に、ステップＳ１３７、およびステップＳ１５１～Ｓ１５４の処理と同じである。ただし、ステップＳ１６１～Ｓ１６５では、集合UG-childから要素Ｃd-i が抽出され、さらに、その「親資源Ｐrnt-i 」が抽出される。

【０１４７】ステップＳ１６６では、要素Cd-i に対応するフラグメント等価クラスＯＩＦHelを生成する。ステップＳ１６７は「親資源Ｐrnt-i 」の各要素についてステップＳ１６８～Ｓ１７２を実行するための処理である。ステップＳ１６８では、「親資源Ｐrnt-i 」に属する要素Ｐi-k を抽出する。ステップＳ１６９では、要素Ｐi-k が属している非フラグメント等価クラスHekを探す。ステップＳ１７０では、非フラグメント等価クラスHekの各要素をフラグメント等価クラスＯＩＦHelに加える。ステップＳ１７１では、非フラグメント等価クラスHekの各要素のためのフラグメント等価フィールドに「フラグメント等価クラスＯＩＦHel」を設定する。そして、ステップＳ１７２において、各要素のための非フラグメント等価フィールドから非フラグメント等価クラスHekを削除する。

【０１４８】図４２は、「名称等価」を検出する処理のフローチャートである。このフローチャートは、図４に示したフローチャートのステップＳ１３の実施例である。ステップＳ１８１では、レジスタリストSr をスキャンする。ステップＳ１８２は、レジスタリストSr の各要素についてステップＳ１８３～Ｓ１８９の処理を実行するための処理である。ステップＳ１８３では、レジスタリストSr から要素（レジスタ）Sr-l を抽出する。

【０１４９】ステップＳ１８４～Ｓ１８６では、抽出した要素Sr-l の名称と、出力ポートリストSo の各要素の名称とを比較する。そして、出力ポートリストSo の中に要素Sr-l の名称と同じ名称を持つ要素So-j が存在している場合には、ステップＳ１８７において、その要素So-j が属する非フラグメント等価クラスHekを探す。ステップＳ１８８では、要素Sr-l のための非フラグメント等価クラスフィールドに「非フラグメント等価クラスHek」を設定する。そして、ステップＳ１８９において、要素Sr-l を非フラグメント等価クラスHekの要素として追加する。

【０１５０】上記処理により、あるレジスタとそのレジスタと同じ名称が付与されている出力ポートとが、「名称等価」とみなされる。図４３は、「フラグメントエッジ」を検出する処理のフローチャートである。この処理では、各データ転送エッジが「フラグメントエッジ」または「非フラグメントエッジ」のいずれに属するのかを判断するために有用な情報を生成する。なお、このフローチャートは、図４に示したフローチャートのステップＳ１４の実施例である。

【０１５１】ステップＳ１９１では、データ転送エッジリストEdtをスキャンする。ステップＳ１９２は、データ転送エッジリストEdtの各要素についてステップＳ１９３～Ｓ２００を実行するための処理である。ステップＳ１９３では、データ転送エッジリストEdtから要素（エッジ）Ｅj を抽出する。このとき、エッジＥj に対応するデータ転送の転送元および転送先を認識する。

【０１５２】ステップＳ１９４では、エッジＥj に対応するデータ転送の転送先が「フラグメント等価」であるか否かを調べる。また、その転送先が「フラグメント等価」であれば、ステップＳ１９５において、エッジＥj に対応するデータ転送の転送元が「フラグメント等価」であるか否かを調べる。そして、その転送先および転送元が共に「フラグメント等価」であった場合には、ステップＳ１９６において両フラグメントフラグをセットし、転送先のみが「フラグメント等価」であった場合には、ステップＳ１９７において転送先フラグメントフラグをセットする。一方、転送先が「フラグメント等価」でなかった場合には、ステップＳ１９８において、エッジＥj に対応するデータ転送の転送元が「フラグメント等価」であるか否かを調べる。そして、転送元のみが「フラグメント等価」であった場合には、ステップＳ１９９において、転送元フラグメントフラグをセットし、転送先および転送元がいずれも「フラグメント等価」でなかった場合には、ステップＳ２００において非フラグメントフラグをセットする。

【０１５３】上記処理により、各データ転送に対応するエッジが、両フラグメントエッジ、転送先フラグメントエッジ、転送元フラグメントエッジ、または非フラグメントエッジに分類される。

【０１５４】図４４～図４８は、「等価エッジ」を検出する処理のフローチャートである。このフローチャートは、図４に示したフローチャートのステップＳ１５の実施例である。

【０１５５】ステップＳ２１１では、データ転送エッジリストEdtをスキャンする、ステップＳ２１２は、データ転送エッジリストEdtの各要素についてステップＳ２１３以降の処理を実行するための処理である。ステップＳ２１３では、データ転送エッジリストEdtから要素（エッジ）Ｅj を抽出する。このとき、エッジＥj に対応するデータ転送の転送元および転送先を認識する。

【０１５６】ステップＳ２１４では、エッジＥj が転送元フラグメントか否か調べる。エッジＥj が転送元フラグメントであればステップＳ２１５へ進み、そうでない場合は、ステップＳ２４１へ進む。ステップＳ２１５では、転送元のフラグメント等価を「ＦHej」とする。ステップＳ２１６では、フラグメント等価ＦHejに属する要素をスキャンする。ステップＳ２１７は、フラグメント等価ＦHejに属する各要素についてステップＳ２１８

～Ｓ２２４を実行するための処理である。

【０１５７】ステップＳ２１８では、フラグメント等価ＦＨejから要素（ポート等）Ni を抽出する。ステップＳ２１９では、要素Ni を転送元ノードまたは転送先ノードとして含むすべてのエッジを抽出する。ステップＳ２２０では、抽出したエッジのリストをスキャンする。ステップＳ２２１は、ステップＳ２１９で抽出した各エッジについてステップＳ２２２～Ｓ２２４を実行するための処理である。ステップＳ２２２では、上記抽出したエッジのリストから要素（エッジ）Ek を抽出する。このとき、エッジEk に対応するデータ転送の転送先ノードを「Head-k」とする。ステップＳ２２３では、この転送先ノードHead-kが、ステップＳ２１３で抽出した転送先と同じであるか否かを調べる。そして、それらが互いに同じ場合は、ステップＳ２２４において、エッジＥｊの等価エッジの集合Ｅqv-Tail-Ejにエッジ Ek を追加する。

【０１５８】フラグメント等価ＦＨejに属するすべての要素についてステップＳ２１８～Ｓ２２４の処理を実行すると、ステップＳ２３１において、集合Ｅqv-Tail-Ej に属するエッジのノードのビット幅ベクトルに基づいてインターバルグラフＩＧを作成する。

【０１５９】ここで、図４９～図５１を参照しながら、インターバルグラフについて説明する。ここでは、図４９に示すデータ転送グラフが生成されているものとする。また、図４９に示すノード間のデータ転送は、図５０に示すように定義されているものとする。そして、各エッジ毎に、ガード条件に対応する制御信号、転送先ノードにおいて使用されるビット、および転送先ノードにおいてビットされるビットが定義されているものとする。この定義によれば、例えば、エッジＲＴは、制御信号G3 が与えられたときに、ノードＲの第７～０ビットがノードＴの第７～０ビットへ転送されることを表している。なお、各データ転送は、T1 ～T10を用いて表わされている。

【０１６０】データ転送グラフからインターバルグラフを作成する場合、まず、各データ転送を点（ドット）で表す。そして、データ転送毎に、転送先ノードにおいて使用すべきビットがオーバラップするデータ転送を抽出し、それらを接続する。

【０１６１】例えば、データ転送T1 に注目する。データ転送T1 によるデータは、ノードＴの第７～０ビットに書き込まれる。このとき、データ転送T2 によるデータはノードＴの第３～０ビットに書き込まれる。したがって、データ転送T1 およびデータ転送T2 により転送されるデータは、ノードTにおいてオーバラップすることになる。同様に、データ転送T3 、T8 、T9 及びT10によるデータも、ノードTにおいてデータ転送T1 によるデータとオーバラップする。この場合、データ転送T1 を表す点は、図５１に示すように、データ転送T2 、T3 、T8 、T9 およびT10を表す点と接続される。

【０１６２】この後、他のデータ転送についても同様の方法を実行することにより、図５１に示すインターバルグラフが得られる。フローチャートの説明に戻る。ステップＳ２３２では、インターバルグラフＩＧにおいて、部品（図５１における「点」）同士の接続を調べる。ステップＳ２３３では、非単体部品の集合ＮＳＣＣを抽出する。各集合ＮＳＣＣには、インターバルグラフにおいて互いにメッシュ状に接続された点に対応する複数のデータ転送から構成される。例えば、図５１において、点T1 、T2 、T3 、T10は、互いにメッシュ状に接続されており、これらの点にそれぞれ対応するデータ転送は１つの集合ＮＳＣＣに属する。ステップＳ２３４では、単体部品の集合ＳＣＣを抽出する。集合ＳＣＣには、いずれの集合ＮＳＣＣにも属していないデータ転送が属することになる。

【０１６３】ステップＳ２３５では、各集合ＮＳＣＣに属し、且つ集合Ｅqv-Tail-Ejに含まれるエッジから独立サブセットＩnd-Eqv-Tail-Ejを作成する。ステップＳ２３６では、集合Ｅqv-Tail-Ejから独立サブセットＩnd-Eqv-Tail-Ejを削除する。ステップＳ２３７では、集合ＮＳＣＣに基づいて、集合Ｅqv-Tail-Ejを１以上のサブグループに分割する。そして、ステップＳ２３８において、各サブグループごとにエッジリストＥfinal を得る。そして、ステップＳ２３９において、各サブグループの集合Ｅqv-Tail-Ejをエッジリスト Efinal に追加する。

【０１６４】エッジＥj が転送元フラグメントでなかった場合（Ｓ２１４：Ｎo）は、ステップＳ２４１において、エッジＥj が転送先フラグメントまたは両フラグメントであるか否かを調べる。エッジＥj が転送先フラグメントまたは両フラグメントであればステップＳ２４２へ進み、そうでない場合は、ステップＳ２４９においてそのエッジＥj を単体ブロックとしてエッジリストＥfinal に加えた後にステップＳ２１２に戻る。

【０１６５】ステップＳ２４２では、非フラグメントの親資源UF-Parent-j が見つかるまでエッジＥj に対応するデータ転送の転送先の親資源フィールドをトラバースしていく。ステップＳ２４３では、親資源UF-Parent-j が属する非フラグメント等価クラスを「ＵＨＰej」とする。ステップＳ２４４では、非フラグメント等価クラスＵＨＰejをスキャンする。ステップＳ２４５は、非フラグメント等価クラスＵＨＰejに属する各要素についてステップＳ２４６～Ｓ２４８を実行するための処理である。ステップＳ２４６では、非フラグメント等価クラスＵＨＰejから要素（ノード）Ni を抽出する。ステップＳ２４７では、ノードNi を転送先ノードとして含むすべてのエッジを抽出する。ステップＳ２４８では、抽出された各エッジを等価エッジの集合Ｅqv-Head-Ejに追加

する。

【０１６６】ステップＳ２５１では、非フラグメント等価クラスＵＨＰejから抽出されたノードＮi がフラグメント等価か否かを調べる。ノードＮi がフラグメント等価であればステップＳ２５２へ進み、そうでなければステップＳ２４５に戻る。ステップＳ２５２では、ノードＮi のフラグメント等価クラスを「ＦHej」とする。ステップＳ２５３では、フラグメント等価クラスＦHejをスキャンする。ステップＳ２５４は、フラグメント等価クラスＦHejに属する各要素についてステップＳ２５５～Ｓ２５７を実行するための処理である。ステップＳ２５５では、フラグメント等価クラスＦHejからノードＮk を抽出する。ステップＳ２５６では、転送先ノードとしてノードNk を含むすべてのエッジを検出する。そして、ステップＳ２５７において、抽出した各エッジを等価エッジの集合Ｅ-Head-Ejに追加する。

【０１６７】各ノードについてステップＳ２４６～Ｓ２４８を実行すると（ステップＳ２４５：Ｙｅｓ）、続いてステップＳ２６１～Ｓ２６８を実行する。ステップＳ２６１～Ｓ２６８は、基本的に、ステップＳ２３１～Ｓ２３７、およびＳ２３９と同じである。ただし、ステップＳ２６１～Ｓ２６８では、集合Ｅqv-Head-Ejに属するエッジに係わるインターバルグラフＩＧが作成され、そのグラフを利用してエッジリストＥfinal が得られる。

【０１６８】図５２～図５５は、「レジスタファイル」を検出する処理のフローチャートである。このフローチャートは、図４に示したフローチャートのステップＳ１６の実施例である。

【０１６９】ステップＳ２７１では、データ転送エッジリストＥdtをスキャンする。ステップＳ２７２は、データ転送エッジリストＥdtに属する各要素についてステップＳ２７３～Ｓ２８４を実行するための処理である。ステップＳ２７３では、データ転送エッジリストＥdtから要素（エッジ）Ｅj を抽出する。また、エッジＥj に対応するデータ転送の転送元Sourceおよび転送先Sinkを検出する。

【０１７０】ステップＳ２７４では、エッジＥj の転送元または転送先がレジスタであるか否かを調べる。転送元または転送先がレジスタであれば、ステップＳ２７５へ進み、そうでない場合はステップＳ２７２に戻る。ステップＳ２７５では、エッジＥj のガード表現のサポートセットを構成する制御信号の集合ＳＳＳ－Ｅj を作成する。ステップＳ２７６では、検出した転送先がレジスタであるか否かを調べる。転送先がレジスタであればステップＳ２７７へ進み、そうでない場合はステップＳ２７７～２８０をスキップする。

【０１７１】ステップＳ２７７では、制御信号の集合ＳＳＳ－Ｅj を転送先に関連づける。ステップＳ２７８では、転送先がレジスタであるデータ転送のガード表現に含まれる制御信号の集合Sink-Support-Setを得る。ステップＳ２７９では、制御信号の集合ＳＳＳ－Ｅj を制御信号の集合Sink-Support-Setに関連づける。ステップＳ２８０では、転送先を転送先レジスタの集合Sink-Registorsに追加する。

【０１７２】ステップＳ２８１では、検出した転送元がレジスタであるか否かを調べる。転送元がレジスタであればステップＳ２８２へ進み、そうでない場合は、ステップＳ２７２に戻る。ステップＳ２８２では、制御信号の集合ＳＳＳ－Ｅj を転送元に関連づける。ステップＳ２８３では、制御信号の集合ＳＳＳ－Ｅj を制御信号の集合Source-Support-Setに関連づける。そして、ステップＳ２８４において、転送元を転送元レジスタの集合Source-Registorsに追加する。

【０１７３】すべてのエッジについてステップＳ２７３～Ｓ２８４の処理を実行すると、ステップＳ２９１において、制御信号の集合Source-Support-Setの結合体U-Source-Sets を作成する。また、ステップＳ２９２において、制御信号の集合Sink-Support-Setの結合体U-Sink-Sets を作成する。ステップＳ２９３では、結合体U-Source-Sets に属するユニークな要素から構成されるランダムトータルオーダTo-U-Source-Setsを作成する。また、ステップＳ２９４では、結合体U-Sink-Sets に属するユニークな要素から構成されるランダムトータルオーダTo-U-Sink-Setsを作成する。

【０１７４】ステップＳ２９５では、Sink-Support-Setをスキャンする。ステップＳ２９６は、Sink-Support-Setに属するすべてのセットについてステップＳ２９７～Ｓ３００を実行するための処理である。ステップＳ２９７では、Sink-Support-SetからＳＳＳ－Ｅj を抽出する。ステップＳ２９８では、To-U-Sink-Setsに基づいてＳＳＳ－Ｅj のビットベクトル表現ＢＶ－ＳＳＳ－Ｅj を作成する。ここで、ＳＳＳ－Ｅj の要素に対応するエントリに「1」を設定し、他の要素に「0」を設定する。ステップＳ２９９では、ＢＶ－ＳＳＳ－Ｅj をＳＳＳ－Ｅj に関連づける。そして、ステップＳ３００において、ＢＶ－ＳＳＳ－Ｅj をＳＳＳ－Ｅj に追加する。このとき、ＢＶ-Sink-Support-Setsは、Sink-Support-Setのビットベクトル表現の集合である。

【０１７５】すべてのセットについてステップＳ２９７～Ｓ３００の処理を実行すると、ステップＳ３１１へ進む。ステップＳ３１１は、Source-Support-Setに属するすべての集合についてステップＳ３１２～Ｓ３１５を実行するための処理である。ステップＳ３１２～Ｓ３１５は、基本的に上述したステップＳ２９７～Ｓ３００と同じである。

【０１７６】ステップＳ３１６では、ＢＶ-Sink-Support-Set のベクトルリストをスキャンする。ステップＳ３１７は、ベクトルリストに属する各ベクトルについてステップＳ３１８～Ｓ３２４を実行するための処理である。ステップＳ３１８では、ベクトルリストからＢＶ－

ＳＳＳ－Ｅｊ を抽出する。ステップＳ３１９では、ＢＶ-Sink-Support-Set のベクトルリストを再スキャンする。ステップＳ３２０は、ベクトルリストに属する各ベクトルについてステップＳ３２１～Ｓ３２３を実行するための処理である。ステップＳ３２１では、ベクトルリストからＢＶ－ＳＳＳ－Ek を抽出する。ステップＳ３２２では、ＢＶ－ＳＳＳ－Ｅｊ とＢＶ－ＳＳＳ－Ek が同じであるか否かを調べる。これらが互いに同じであれば、ステップＳ３２３において、ＢＶ－ＳＳＳ－Ek に対応する転送先レジスタReg-sink-kを検出する。そして、すべてのベクトルについてステップＳ３２１～Ｓ３２３の処理を実行すると、ステップＳ３２４において、転送先レジスタReg-sink-kを転送先レジスタファイルの集合Sink-Reg-Filesに追加する。

【０１７７】ステップＳ３３１～Ｓ３３９は、基本的にステップＳ３１６～Ｓ３２４の処理と同じである。ただし、ステップＳ３３１～Ｓ３３９では、ＢＶ-Sink-Support-Set のベクトルリストに属する各要素について処理が行われ、所定の転送元レジスタが転送元レジスタファイルの集合Source-Reg-Filesに追加される。

【０１７８】ステップＳ３４１では、転送先レジスタファイルの集合Sink-Reg-Filesのリストをスキャンする。ステップＳ３４２は、そのリストに属する各集合についてステップＳ３４３～Ｓ３４８を実行するための処理に相当する。ステップＳ３４３では、上記のリストから集合Sink-Reg-File-k を抽出する。ステップＳ３４５では、すべての集合が処理されたか否かを調べる。未処理のセットがあれば、ステップＳ３４６において、集合Source-Reg-File-l を抽出する。ステップＳ３４７では、集合Sink-Reg-File-k と集合Source-Reg-File-l とが同一であるか否かを調べる。そして、それらが互いに同じであれば、ステップＳ３４８において、集合Source-Reg-File-k を集合Reg-Files に追加する。上記処理により、レジスタファイルに属するレジスタが得られる。

【０１７９】このように、図３２～図３５に示すフローチャートに処理により生成されたデータ転送グラフは、上述のフローチャートの処理により最適化される。そして、その最適化の結果は、エッジリストＥfinal である。

【０１８０】図５７は、最適化されたデータ転送グラフから隣接ノードリストを作成する処理のフローチャートである。なお、このフローチャートは、図１９に示したフローチャートのステップＳ２１の実施例である。

【０１８１】ステップＳ３５１では、エッジリストEfinal をスキャンする。ステップＳ３５２は、エッジリストEfinal に属する各サブグループについてステップＳ３５３～Ｓ３６１を実行するための処理に相当する。ステップＳ３５３では、エッジリストEfinal からサブグループＰi を抽出する。

【０１８２】ステップＳ３５４は、サブグループＰi に属する各エッジについてステップＳ３５５～Ｓ３６１の処理を実行するための処理に相当する。ステップＳ３５５では、サブグループＰi からエッジＥｊ を抽出する。このとき、エッジＥｊ の転送先ノードおよび転送元ノードが属する非フラグメント等価クラスにそれぞれ対応する等価クラスHead-Hejおよび等価クラスTail-Hejを抽出する。ステップＳ３５６では、非フラグメント等価クラスHead-Hejが隣接ノードリストのノードセットＮの要素であるか否かを調べる。そして、非フラグメント等価クラスHead-HejがノードセットＮの要素でなかった場合には、ステップＳ３５７においてそれをノードセットＮに追加する。同様に、ステップＳ３５８では、非フラグメント等価クラスTail-HejがノードセットＮの要素であるか否か調べる。そして、非フラグメント等価クラスTail-HejがノードセットＮの要素でなかった場合には、ステップＳ３５９においてそれをノードセットＮに追加する。

【０１８３】ステップＳ３６０では、隣接ノードリストのノードセットＮにおける非フラグメント等価クラスHead-Hejの隣接リストに非フラグメント等価クラスTail-Hejを追加する。また、非フラグメント等価クラスHead-Hejの入力エッジ数をインクリメントする。一方、ステップＳ３６１では、ノードセットＮにおける非フラグメント等価クラスTail-Hejの隣接リストに非フラグメント等価クラスHead-Hejを追加する。そして、非フラグメント等価クラスTail-Hejの入力エッジ数をインクリメントする。

【０１８４】図５８～図６０は、検証ツールに入力すべきプロパティスクリプトを生成する処理のフローチャートである。なお、このフローチャートは、図１９に示したフローチャートのステップＳ２２の実施例である。

【０１８５】ステップＳ３７１では、隣接ノードリストのノードセットＮをスキャンする。ステップＳ３７２は、ノードセットＮに属する各ノードについてステップＳ３７３以降の処理を実行するための処理である。ステップＳ３７３では、ノードセットからノードＮｊ を抽出する。

【０１８６】ステップＳ３７４では、ノードＮｊ のファンインエッジ数が１以上であるか否かを調べる。ファンインエッジ数が１以上であればステップＳ３７５へ進み、そうでない場合はステップＳ３７２に戻る。ステップＳ３７５では、転送先ノードがノードＮｊ であるエッジ（ノードＮｊ へのファンインエッジ）の集合Di-Arcs-into-Njを得る。ステップＳ３７６では、ノードＮｊ のファンアウトエッジ数が１以上であるか否かを調べる。出力エッジ数が１以上であればステップＳ３７７へ進み、そうでない場合はステップＳ３７７をスキップする。

【０１８７】ステップＳ３７７では、転送元ノードがノードＮｊ であるエッジ（ノードＮｊからのファンアウト

エッジ）の集合Di-Arcs-out-of-Njを得る。ステップＳ３７８では、ノードNj がレジスタであるか否かを調べる。ノードNj がレジスタであればステップＳ３７９へ進み、そうでない場合はステップＳ４１１へ進む。ステップＳ３７９では、集合Di-Arcs-into-Njに属するファンインエッジが存在するか否かを調べる。集合Di-Arcs-into-Njに属するファンインエッジが存在する場合はステップＳ３８０において、それらのファンインエッジ同士の資源競合プロパティを生成する。そうでない場合はステップＳ３７２に戻る。

【０１８８】ステップＳ３８１では、集合Di-Arcs-out-of-Njに属するファンアウトエッジが存在するか否かを調べる。集合Di-Arcs-out-of-Njに属するファンアウトエッジが存在する場合は、ステップＳ３８０において、集合Di-Arcs-into-Njと集合Di-Arcs-out-of-Njとの積ＰＳ１を生成する。ＰＳ１を「Em 、En 」と表す。なお、「Em 」及び「En 」は、それぞれ集合Di-Arcs-into-Njおよび集合Di-Arcs-out-of-Njの要素である。一方、集合Di-Arcs-out-of-Njに属するファンアウトエッジが存在しない場合は、ステップＳ３７２に戻る。

【０１８９】ステップＳ３９１では、ＰＳ１をスキャンする。ステップＳ３９２は、ＰＳ１の各要素についてステップＳ３９３～Ｓ３９５の処理を実行するための処理に相当する。ステップＳ３９３では、ＰＳ１から要素ＰＳ-k（Ea 、Eb ）を抽出する。ステップＳ３９４では、集合Di-Arcs-out-of-Njの要素Eb の処理タイプが順次転送であるか否かを調べる。そして、処理タイプが順次転送であれば、ステップＳ３９５において、クロック歪みによるエラーをチェックするための資源競合プロパティを生成する。

【０１９０】ステップＳ３９６では、集合Di-Arcs-into-Njと集合Di-Arcs-into-Njとの積ＰＳ２を生成する。続いて、ステップＳ３９７では、エッジEj のガード表現を「Gi 」としたときに、集合Di-Arcs-out-of-Njに属する各ファンアウトエッジEi に対して論理式No-Read=-(OR(Gi)) を生成する。ステップＳ３９８では、ＰＳ２をスキャンする。ステップＳ３９９は、ＰＳ２に属する各要素について処理４００およびＳ４０１を実行するための処理に相当する。

【０１９１】ステップＳ４００では、ＰＳ２からＰＳ２-k=（Ec 、Ed ）を抽出する。そして、ステップＳ４０１において、レジスタ漏れプロパティEF(Gl"EX(E(No-ReadUGm)))を生成する。

【０１９２】ノードNj がレジスタでなかった場合（ステップＳ３７８：Ｎｏ）は、ステップＳ４１１において、そのノードNj が出力ポートであるか否かを調べる。ノードNj が出力ポートであれば、ステップＳ４１２において、集合Di-Arcs-into-Njに要素が存在するか否かを調べる。すなわち、その出力ポートへのファンインエッジが存在するか否かを調べる。ファンインエッジがある場合には、ステップＳ４１３において、それらエッジ同士のすべての組合せに対して資源競合プロパティを生成する。

【０１９３】ステップＳ４１４では、ノードNj が入力ポートであるか否か調べる。ノードNj が入力ポートであれば、ステップＳ４１５において、集合Di-Arcs-into-Njおよび集合Di-Arcs-into-Njの結合体Union-Arcs-of-Njを生成する。そして、ステップＳ４１６において、Union-Arcs-of-NjとUnion-Arcs-of-Njとの積ＰＳ３を生成する。

【０１９４】ステップＳ４１７では、ＰＳ３をスキャンする。ステップＳ４１８は、ＰＳ３に属する各要素についてステップＳ４１９～Ｓ４２１の処理を実行するための処理に相当する。ステップＳ４１９では、ＰＳ３からＰＳ３-k=（Ee 、Eg ）を抽出する。ステップＳ４２０では、エッジEe とエッジEg が同じものであるか調べる。それらのエッジが同じであった場合は、ステップＳ４２１において、資源競合プロパティを生成する。

【０１９５】上述したプロパティを生成する機能は、コンピュータを用いて上述のフローチャートに示した処理を記述したプログラムを実行することにより実現される。そのプログラムを実行するコンピュータ１００のブロック図を図６１に示す。

【０１９６】ＣＰＵ１０１は、上述のフローチャートに示した処理を記述したプログラムを記憶装置１０２からメモリ１０３にロードして実行する。記憶装置１０２は、たとえばハードディスクであり、上記プログラムを格納する。一方、メモリ１０３は、例えば半導体メモリであり、ＣＰＵ１０１の作業領域として使用される。

【０１９７】記録媒体ドライバ１０４は、ＣＰＵ１０１の指示に従って可搬性記録媒体１０５にアクセスする。可搬性記録媒体１０５は、例えば、半導体デバイス（ＰＣカード等）、磁気的作用により情報が入出力される媒体（フロッピーディスク、磁気テープなど）、光学的作用により情報が入出力される媒体（光ディスクなど）を含む。通信制御装置１０６は、ＣＰＵ１０１の指示に従って網との間でデータを送受信する。

【０１９８】図２３は、本発明に係わるソフトウェアプログラムなどの提供方法を説明する図である。本発明に係わるプログラムは、例えば、以下の３つの方法の中の任意の方法により提供される。

【０１９９】(a) コンピュータ１００にインストールされて提供される。この場合、プログラム等は、たとえば、出荷前にプレインストールされる。

(b) 可搬性記録媒体に格納されて提供される。この場合、可搬性記録媒体１０５に格納されているプログラム等は、基本的に、記録媒体ドライバ１０４を介して記憶装置１０２にインストールされる。

【０２００】(c) 網上のサーバから提供される。この場合、基本的には、コンピュータ１００がサーバに格納さ

れているプログラム等をダウンロードすることによってそのプログラム等を取得する。網は、無線網を含む。

【０２０１】なお、上述の実施例では、検証すべきプロパティとして、資源競合およびレジスタ漏れを採り上げているが、本発明が生成するプロパティはこれに限定されるものではない。

【０２０２】また、データ資源としてレジスタ、ポート、バスを採り上げているが、本発明が対象とするの資源はこれに限定されるものではない。

【０２０３】

【発明の効果】本発明によれば、ハードウェア記述言語で記述された仕様から、そのハードウェアにおいて資源競合またはレジスタ漏れが発生するか否かを検証するためのプロパティを自動的に生成できる。このため、ハードウェアの設計ミスを早い段階で修正でき、ＩＣ等の開発のための時間およびコストが節約される。また、デバッグ作業が減るので、設計者は、余った労力を他の作業に投入できる。

【図面の簡単な説明】

【図１】本発明の一実施形態のプロパティ作成ツールが仕様される環境を説明する図である。

【図２】プロパティ生成ツールの動作を説明するフローチャートである。

【図３】データ転送グラフの一例である。

【図４】最適化処理のフローチャートである。

【図５】ハードウェアの例である。

【図６】図５に示すハードウェアを記述する階層的に方法を説明する図である。

【図７】「資源の等価」を利用してデータ転送グラフを最適化する例を示す図である。

【図８】「分岐」を説明する図である。

【図９】レジスタファイルを説明する図である。

【図１０】Verilog で記述されたハードウェア仕様の一例（その１）である。

【図１１】Verilog で記述されたハードウェア仕様の一例（その２）である。

【図１２】Verilog で記述されたハードウェア仕様の一例（その３）である。

【図１３】Verilog で記述されたハードウェア仕様の一例（その４）である。

【図１４】Verilog で記述されたハードウェア仕様の一例（その５）である。

【図１５】Verilog で記述されたハードウェア仕様の一例（その６）である。

【図１６】図１０～図１５に示したハードウェア仕様から生成されたデータ転送グラフである。

【図１７】疑似資源を説明する図である。

【図１８】最適化されたデータ転送グラフの例である。

【図１９】プロパティ作成処理のフローチャートである。

【図２０】隣接ノードリストの例である。

【図２１】「資源競合」のプロパティの例である。

【図２２】「レジスタ漏れ」のプロパティの例である。

【図２３】クロックの歪みを考慮した場合の「資源競合」のプロパティの例である。

【図２４】Verilog で記述されたハードウェア仕様の一例（その１）である。

【図２５】Verilog で記述されたハードウェア仕様の一例（その２）である。

【図２６】Verilog で記述されたハードウェア仕様の一例（その３）である。

【図２７】図２４～図２６に示したハードウェア仕様から生成されたデータ転送グラフである。

【図２８】図２７に示すグラフにおけるデータ転送の条件を示す図である。

【図２９】レジスタファイルを検出する処理を説明する図である。

【図３０】最適化されたデータ転送グラフの例である。

【図３１】最適化されたグラフにおいて使用される条件を示す図である。

【図３２】データ転送グラフを生成する方法のフローチャート（その１）である。

【図３３】データ転送グラフを生成する方法のフローチャート（その２）である。

【図３４】データ転送グラフを生成する方法のフローチャート（その３）である。

【図３５】データ転送エッジを生成する処理の詳細フローチャートである。

【図３６】「非フラグメント等価」を検出する処理のフローチャート（その１）である。

【図３７】「非フラグメント等価」を検出する処理のフローチャート（その２）である。

【図３８】「フラグメント等価」を検出する処理のフローチャート（その１）である。

【図３９】「フラグメント等価」を検出する処理のフローチャート（その２）である。

【図４０】「フラグメント等価」を検出する処理のフローチャート（その３）である。

【図４１】「フラグメント等価」を検出する処理のフローチャート（その４）である。

【図４２】「名称等価」を検出する処理のフローチャートである。

【図４３】「分岐エッジ」を検出する処理のフローチャートである。

【図４４】「等価エッジ」を検出する処理のフローチャート（その１）である。

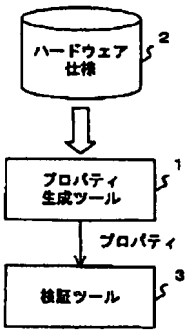
【図４５】「等価エッジ」を検出する処理のフローチャート（その２）である。

【図４６】「等価エッジ」を検出する処理のフローチャート（その３）である。

【図４７】「等価エッジ」を検出する処理のフローチャート（その４）である。
【図４８】「等価エッジ」を検出する処理のフローチャート（その５）である。
【図４９】データ転送グラフの例である。
【図５０】データ転送の定義である。
【図５１】インターバルグラフの例である。
【図５２】「レジスタファイル」を検出する処理のフローチャート（その１）である。
【図５３】「レジスタファイル」を検出する処理のフローチャート（その２）である。
【図５４】「レジスタファイル」を検出する処理のフローチャート（その３）である。
【図５５】「レジスタファイル」を検出する処理のフローチャート（その４）である。
【図５６】「レジスタファイル」を検出する処理のフローチャート（その５）である。
＊【図５７】隣接ノードリストを作成する処理のフローチャートである。
【図５８】プロパティスクリプトを生成する処理のフローチャート（その１）である。
【図５９】プロパティスクリプトを生成する処理のフローチャート（その２）である。
【図６０】プロパティスクリプトを生成する処理のフローチャート（その３）である。
【図６１】本発明の機能を記述したプログラムを実行するコンピュータのブロック図である。
【図６２】本発明に係わるソフトウェアプログラムなどの提供方法を説明する図である。
【符号の説明】
1　プロパティ生成ツール
2　ハードウェア仕様
3　検証ツール

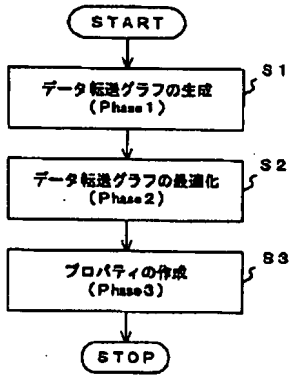
【図１】

本発明の一実施形態のプロパティ作成ツールが仕様される環境を説明する図



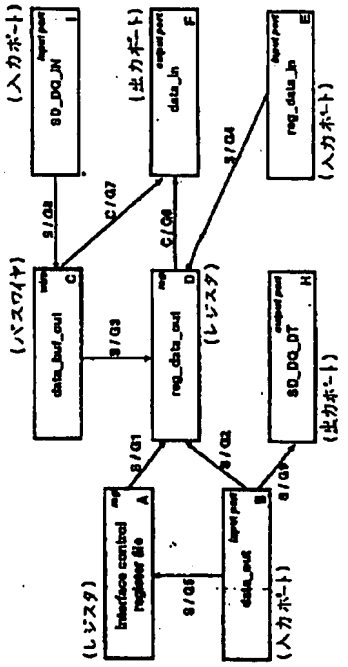
【図２】

プロパティ生成ツールの動作を説明するフローチャート



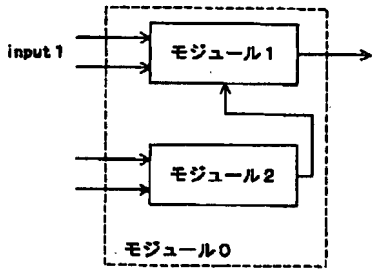
【図３】

データ転送グラフの一例



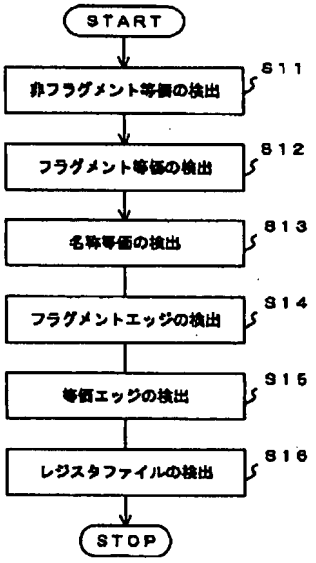
【図５】

ハードウェアの例



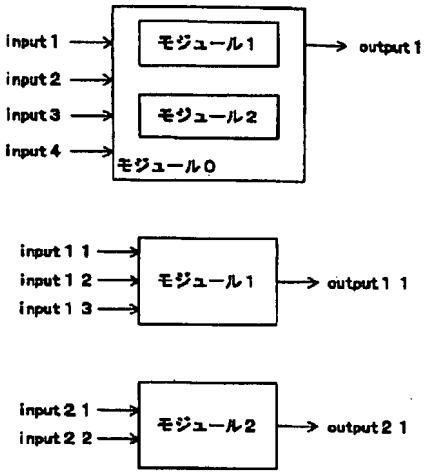
【図４】

最適化処理のフローチャート



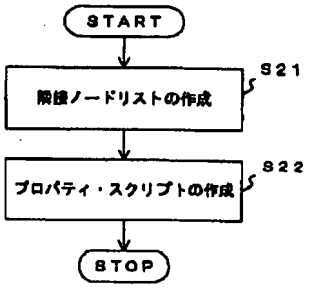
【図６】

図５に示すハードウェアを記述する階層的に方法を説明する図



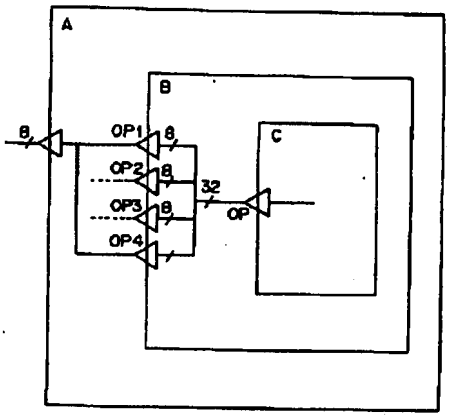
【図１９】

プロパティ作成処理のフローチャート



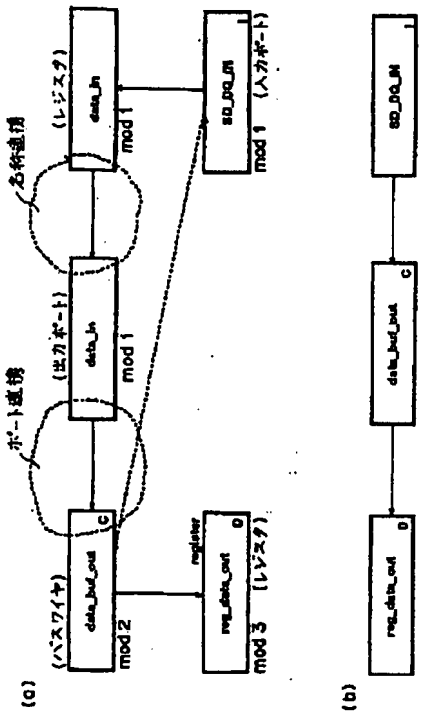
【図８】

「分岐」を説明する図



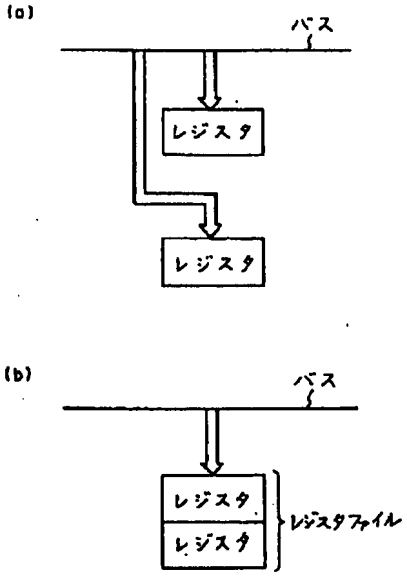
【図７】

「資源の等価」を利用してデータ転送フラグを最適化する例を示す図



【図９】

レジスタファイルを説明する図



【図１０】

Verilogで記述されたハードウェア仕様の一例（その１）

```
module MOD 0 (GLOBAL_CLK, INPUT 1, INPUT 2, OUTPUT 1, OUTPUT 2,
              CTRL 1, CTRL 2, CTRL 3, CTRL 4, CTRL 5, CTRL 6, CTRL 7);

  input         GLOBAL_CLK;
  input  [15:0] INPUT1;
  input  [15:0] INPUT2;
  output [15:0] OUTPUT1;
  output [15:0] OUTPUT2;
  input         CTRL1;
  input         CTRL2;
  input         CTRL3;
  input         CTRL4;
  input         CTRL5;
  input         CTRL6;
  input         CTRL7;

  wire   [15:0] CONEXT;

  MOD1  MOD1_INST_A
    (.GLOBAL_CLK(GLOBAL_CLK),
     .CTRL1(CTRL1),
     .CTRL2(CTRL2),
     ,IPORT1(CONEXT[15:8]),
     .IPORT2(CONEXT[7:0]),
     .IPORT3(INPUT1[15:8]),
     .IPORT4(INPUT1[7:0]),
     .OPORT1(OUTPUT1[15:8]),
     .OPORT2(OUTPUT1[7:0])
    );

  MOD2  MOD2_INST_A
    (.GLOBAL_CLK(GLOBAL_CLK),
     .CTRL3(CTRL3),
     .CTRL4(CTRL4),
     .CTRL5(CTRL5),
     .CTRL6(CTRL6),
     .CTRL7(CTRL7),
     .IPORT5(INPUT2),
     .OPORT3(OUTPUT2),
     .OPORT4(CONEXT)
    );

endmodule
```

①

②

③

【図２０】

隣接ノードリストの例

転送元ノード	転送先ノード
E	L
F	L
F	M
G	M
H	N
I	N
J	O
K	O
Q	V
L	V
M	V
R	V

【図11】

Verilog で記述されたハードウェア仕様の一例 (その2)

```
module MOD1 (GLOBAL_CLK, CTRL1, CTRL2, IPORT1, IPORT2,
             IPORT3, IPORT4, OPORT1, OPORT2);

    input       GLOBAL_CLK;
    input       CTRL1;
    input       CTRL2;
    input [7:0] IPORT1;
    input [7:0] IPORT2;
    input [7:0] IPORT3;
    input [7:0] IPORT4;
    output[7:0] OPORT1;
    output[7:0] OPORT2;

                reg [7:0] REG_A;
C1  :           assign OPORT1 = REG_A;
                reg [7:0] REG_B;
C2  :           assign OPORT2 = REG_B;
```

④ ⑤

【図12】

Verilog で記述されたハードウェア仕様の一例 (その3)

```
S1  :  always @( posedge (GLOBAL_CLK))
       begin
           if (CTRL1)
           begin
               REG_A = IPORT1;
               REG_B = IPORT3;
           end
       end

S2  :  always @( posedge (GLOBAL_CLK))
       begin
           if (CTRL2)
           begin
               REG_A = IPORT2;
               REG_B = IPORT4;
           end
       end

endmodule
```

⑥ ⑦

【図13】

Verilog で記述されたハードウェア仕様の一例 (その4)

```
module MOD2 (GLOBAL_CLK, CTRL3, CTRL4, CTRL5,
             CTRL6, CTRL7, IPORT5, OPORT3, OPORT4);

    input           GLOBAL_CLK;
    input           CTRL3;
    input           CTRL4;
    input           CTRL5;
    input           CTRL6;
    input           CTRL7;
    input  [15:0]   IPORT5;
    output [15:0]   OPORT3;
    output [15:0]   OPORT4;

        reg [15:0] REG_C;

        reg [15:0] REG_D;

        reg [15:0] REG_E;
C3  :   assign OPORT3 = REG_E;

        reg [15:0] REG_F;
C4  :   assign OPORT4 = REG_F;

        initial
        begin
            REG_C = 16'b0000000000000000;
            REG_D = 16'b0000000000000001;
            REG_E = 16'b0000000000000000;
            REG_F = 16'b0000000000000000;
        end
```

⑧ ⑨

【図14】

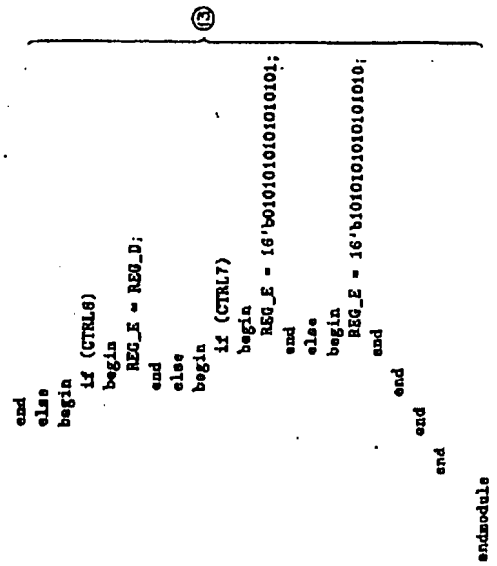
Verilog で記述されたハードウェア仕様の一例 (その5)

```
S3  :  always @( posedge (GLOBAL_CLK))
       begin
           if (CTRL3)
           begin
             REG_C = IPORT5;
           end
           else
           begin
             REG_D = IPORT5;
           end
       end

S4  :  always @( posedge (GLOBAL_CLK))
       begin
           if (CTRL4)
           begin
             REG_C = REG_C + REG_D;
             REG_D = REG_C / REG_D;
           end
       end

S5  :  always @( posedge (GLOBAL_CLK))
       begin
           REG_F = REG_E - REG_D;
       end

S6  :  always @( posedge (GLOBAL_CLK))
       begin
           if (CTRL5)
           begin
             REG_E = REG_C;
```

⑩ ⑪ ⑫ ⑬

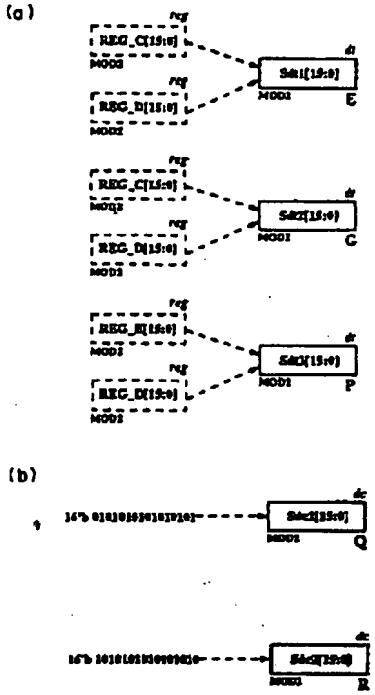
【図１５】

Verilog で記述されたハードウェア仕様の一例（その５）



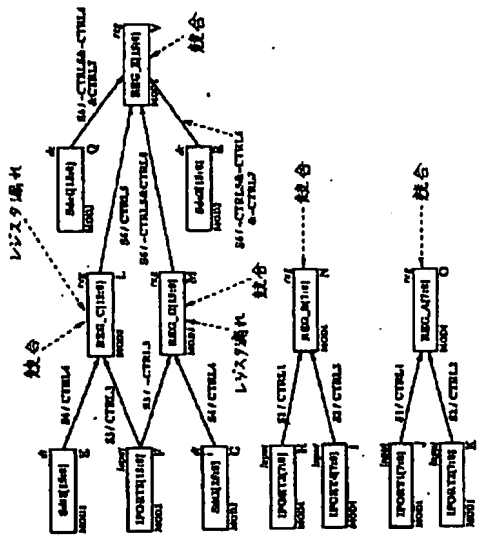
【図１７】

疑似資源を説明する図



【図１８】

最適化されたデータ転送グラフの例



【図２６】

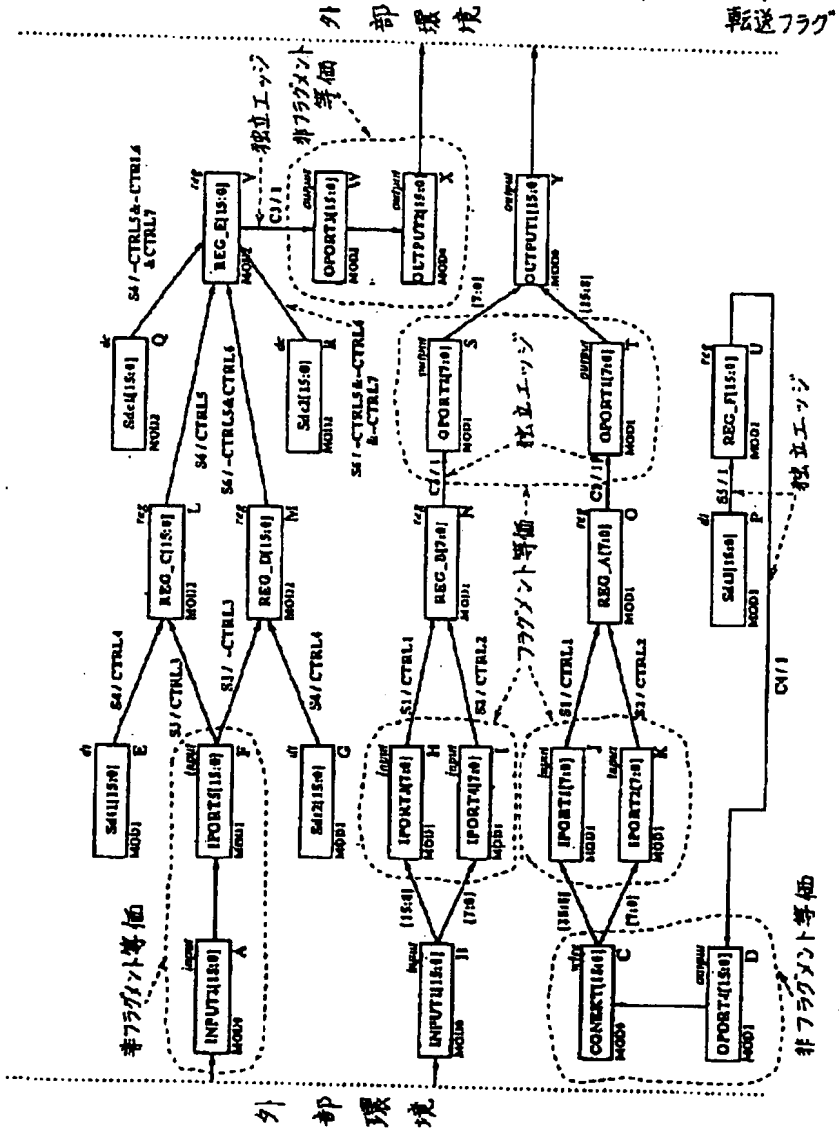
Verilogで記述されたハードウェア仕様の一例
（その３）

```
32  :  always @(posedge GLOBAL_CLK)
       begin
        if(CNT_REG_AREA & COMPARE)
         begin
          REG_ADR =  EXT_ADDR[15:0];
         end
        if(COMPARE)
         begin
          if(WRITE & CNT_REG_AREA)
           begin
            REGA_OUT =  EXT_DATA_OUT;
           end
         end
        else if(!WRITE)
        begin
         if(CNT_REG_ACC)
          begin
           if(REG_ADR[7:5] == 3'b111)
            begin
             case(REG_ADR[4])
              1'b0:          REGA_OUT =     {EXTS,BASIS};
              1'b1:          REGA_OUT =     {MODE,16'h0000};
    default:;
             endcase
            end
           else
           begin
           - REGA_OUT =  REGA_IN;
           end
          end
         end
       end
```

③

【図１６】

図10～図15に示したハードウェア仕様から生成されたデータ



【図２１】

「資源競合」のプロパティの例

```
Resource Contention Properties

Node L :  AG(CTRL4.CTRL3)
Node M :  AG(~CTRL3).CTRL4)
Node N :  AG(CTRL4.CTRL2)
Node O :  AG(CTRL3.CTRL2)
Node V :  AG(!~CTRL5 & ~CTRL6 & CTRL7).CTRL5)
          AG(!~CTRL5 & ~CTRL6 & CTRL7).(~CTRL5 & CTRL6)))
          AG(!~CTRL5 & ~CTRL6 & CTRL7).(~CTRL5 & ~CTRL6 & ~CTRL7)))
          AG(CTRL5.(!~CTRL5 & CTRL6)))
          AG(CTRL5.(!~CTRL5 & ~CTRL6 & ~CTRL7)))
          AG(!~CTRL5 & CTRL6).(~CTRL5 & ~CTRL6 & ~CTRL7)))

          For data transfer edges EL & FL.
          For data transfer edges FM & GM.
          For data transfer edges HN & IN.
          For data transfer edges JO & KO.
          For data transfer edges OV & LV.
          For data transfer edges OV & MV.
          For data transfer edges OV & RV.
          For data transfer edges LV & MV.
          For data transfer edges LV & RV.
          For data transfer edges MV & RV.
```

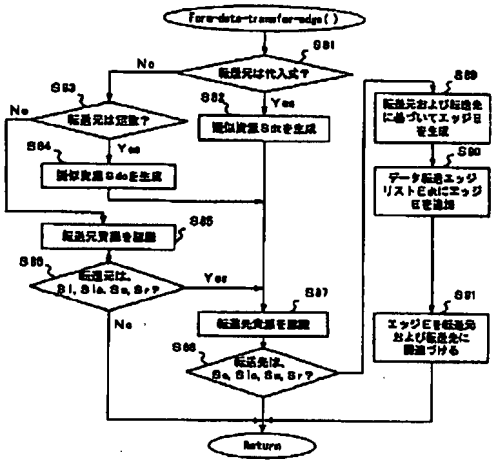
【図２２】

「レジスタ漏れ」のプロパティの例

```
Register Leakage Properties

Node L :  Ro_Read_L = ~CTRL5

          EF(CTRL4 /\ EX(E(~CTRL5 U CTRL4))) ---   For two consecutive writes from E to L
          EF(CTRL3 /\ EX(E(~CTRL5 U CTRL3))) ---   For two consecutive writes from F to L
          EF(CTRL3 /\ EX(E(~CTRL5 U CTRL4))) ---   For a write from F to L followed by a write from E to L
          EF(CTRL4 /\ EX(E(~CTRL5 U CTRL3))) ---   For a write from E to L followed by a write from F to L

Node M :  Ro_Read_M = ~(~CTRL5 & CTRL6)

          EF(~CTRL3 /\ EX(E(Ro_Read_M U ~CTRL3))) ---   For two consecutive writes from F to M
          EF(CTRL4 /\ EX(E(Ro_Read_M U CTRL4))) ---   For two consecutive writes from G to M
          EF(~CTRL3 /\ EX(E(Ro_Read_M U CTRL4))) ---   For a write from F to M followed by a write from G to M
          EF(CTRL4 /\ EX(E(Ro_Read_M U ~CTRL3))) ---   For a write from G to M followed by a write from F to M
```

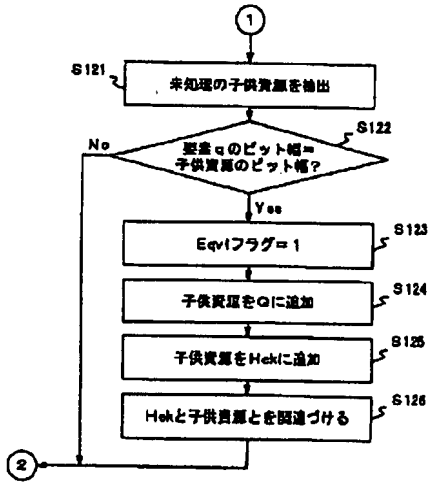
【図３５】

データ転送エッジを生成する処理の詳細フローチャート



【図３７】

「非フラグメント等価」を検出する処理のフローチャート（その２）



【図２３】

クロックの歪みを考慮した場合の「資源競合」のプロパティの例

```
Resource Contention Properties (for clock skew error)
  Node L : AG(CTRL4.CTRL3)
           AG(CTRL3.CTRL3)
  Node M : AG(~CTRL3.(~CTRL5 & CTRL4)))
           AG(CTRL4.(~CTRL5 & CTRL4)))
```

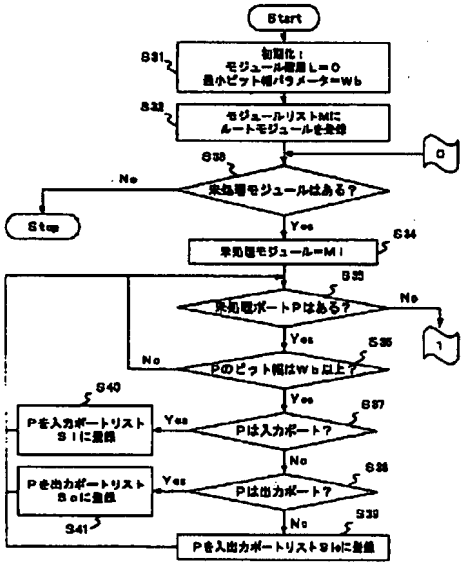
【図２４】

Verilogで記述されたハードウェア仕様の一例（その１）

```
module MODO (GLOBAL_CLK, COMPARE, CNT_REG_ACC, CNT_REG_AREA, WRITE, EXT_ADDR,
             EXT_DATA_OUT, REG4_OUT, REG4_IN);
    input        GLOBAL_CLK;
    input        COMPARE;
    input        CNT_REG_ACC;
    input        CNT_REG_AREA;
    input        WRITE;
    input [15:0] EXT_ADDR;
    input [31:0] EXT_DATA_OUT;
    output[31:0] REG4_OUT;
    input [31:0] REG4_IN;

    reg  [15:0] BASE3,END3,MODE;
    reg  [7:0]  REG_ADR;
    reg  [31:0] REG4_OUT;
```

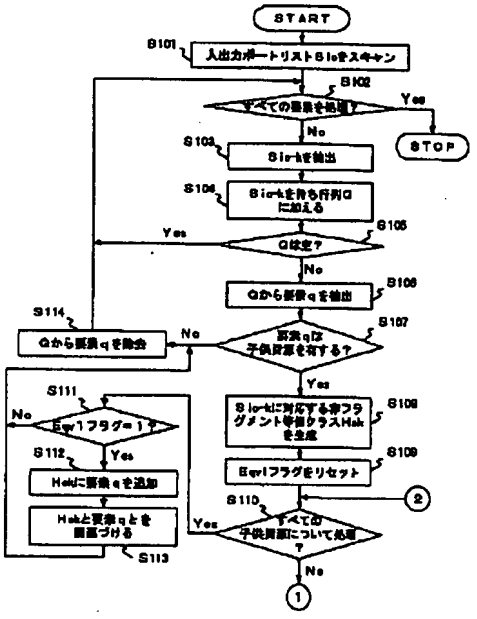
【図３２】

データ転送グラフを生成する方法のフローチャート（その１）



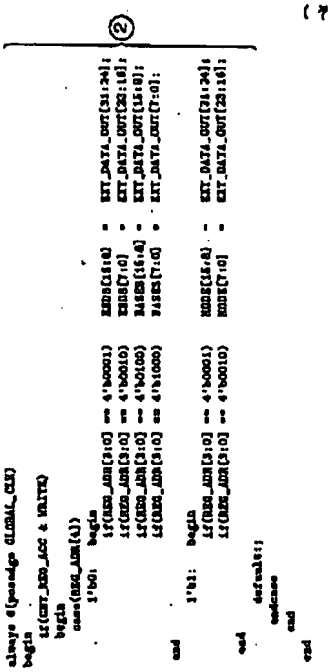
【図３６】

「非フラグメント等価」を抽出する処理のフローチャート（その１）



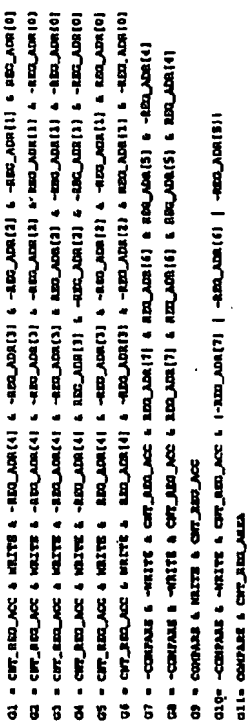
【図２５】

Verilogで記述されたハードウェア仕様の一例（その２）



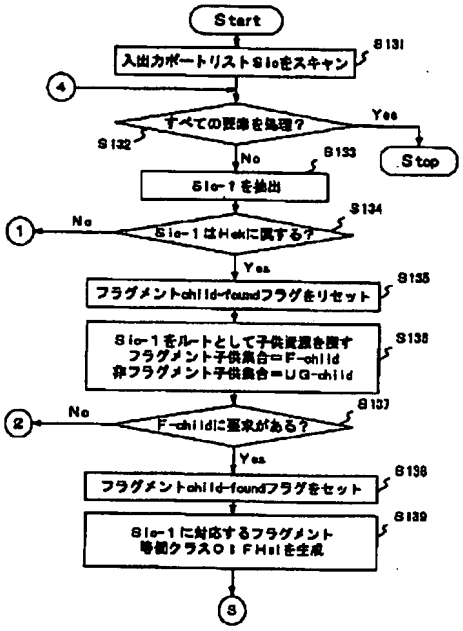
【図２８】

図27に示すグラフにおけるデータ転送の条件を示す図



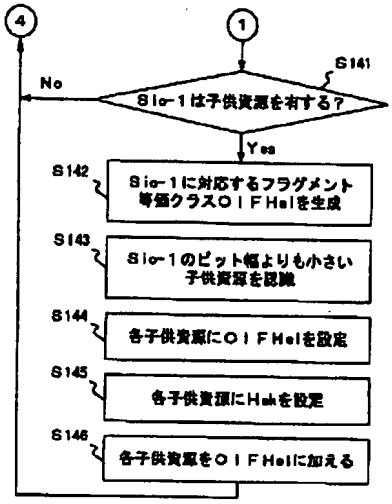
【図３８】

「フラグメント等価」を検出する処理のフローチャート（その１）



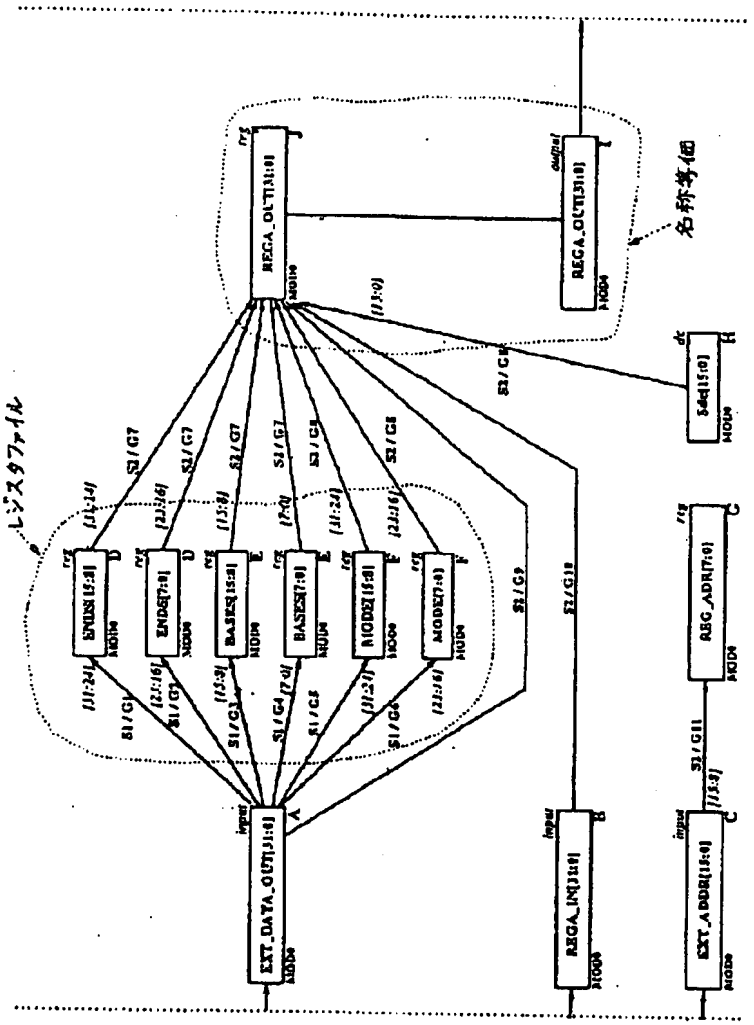
【図３９】

「フラグメント等価」を検出する処理のフローチャート（その２）



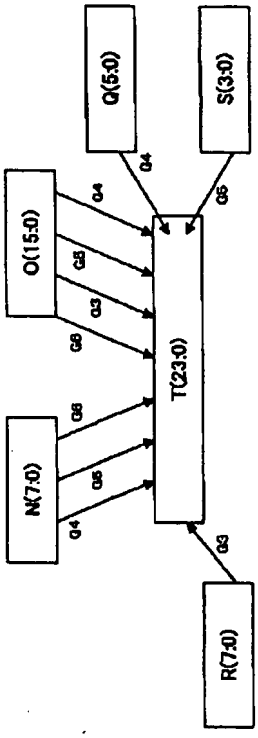
【図２７】

図24 ～ 図26に示したハードウェア仕様から
生成されたデータ転送グラフ



【図４９】

データ転送グラフの例



【図２９】

レジスタファイルを検出する処理を説明する図

```
Register_File_Identification_Steps

Support(C1) = ( CNT_REG_ACC. WRITE. REG_ADR[6]. REG_ADR[3]. REG_ADR[2]. REG_ADR[1]. REG_ADR[0] )
Support(C2) = Support(C2) + Support(C3) + Support(C4) + Support(C5)
Sink_Reg_File = ( ENDS[15:0]. BASES[15:0]. MODE[15:0] )

Support(C7) = ( CONTAINS. WRITE. CNT_REG_ACC. REG_ADR[7]. REG_ADR[6]. REG_ADR[5]. REG_ADR[4] )
Support(C7) = Support(C8)
Source_Reg_File = ( ENDS[15:0]. BASES[15:0]. MODE[15:0] )

As. Source_Reg_File = Sink_Reg_File.
Reg_File = ( ENDS[15:0]. BASES[15:0]. MODE[15:0] )
```

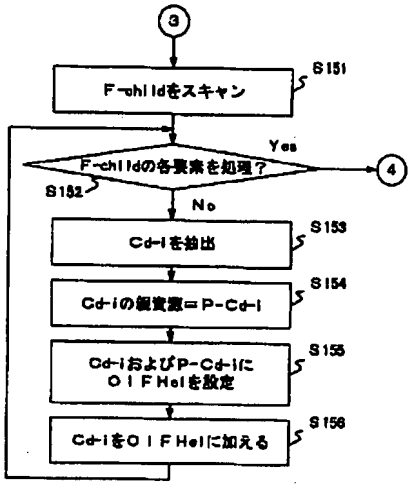
【図３１】

最適化されたグラフにおいて
使用される条件を示す図

```
Mem Check/Enable Expressions In Optimized DTBAG of example2.v

C1' = C1 | C2 | C3 | C4 | C5 | C6

C1' = CNT_REG_ACC & WRITE

C2' = C7 | C8

C3' = ~CONTAINS & ~WRITE & CNT_REG_ACC & REG_ADR[7] & REG_ADR[6] & REG_ADR[5]
```

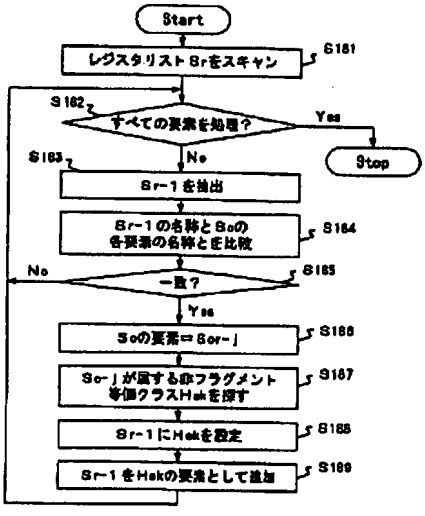
【図４０】

「フラグメント等価」を検出する処理の
フローチャート（その３）



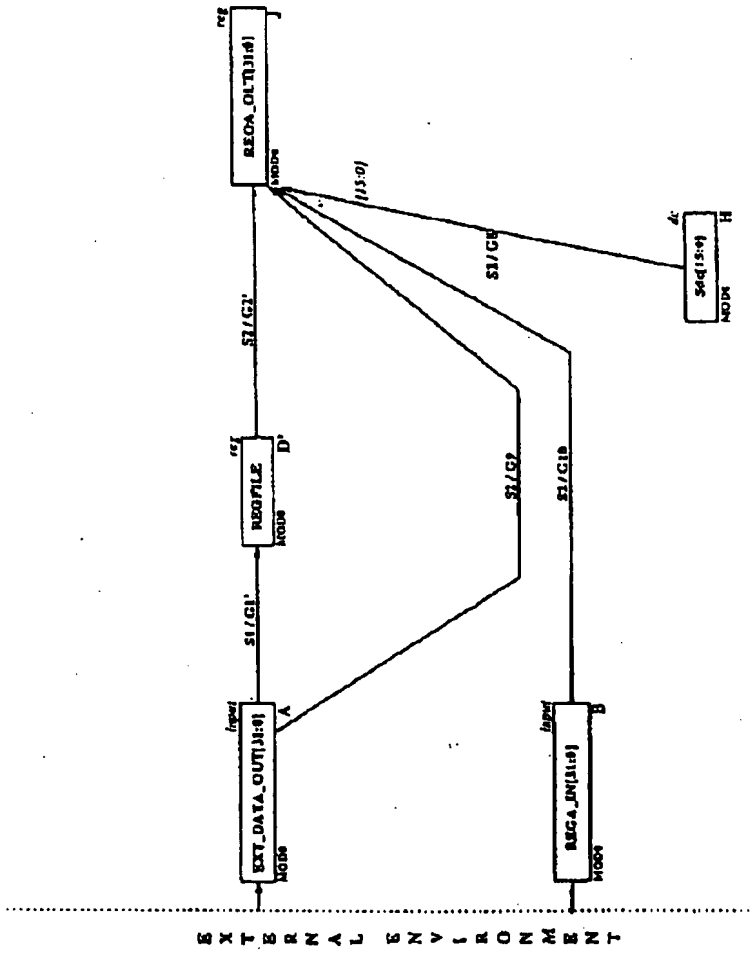
【図４２】

「名称等価」を検出する処理のフローチャート



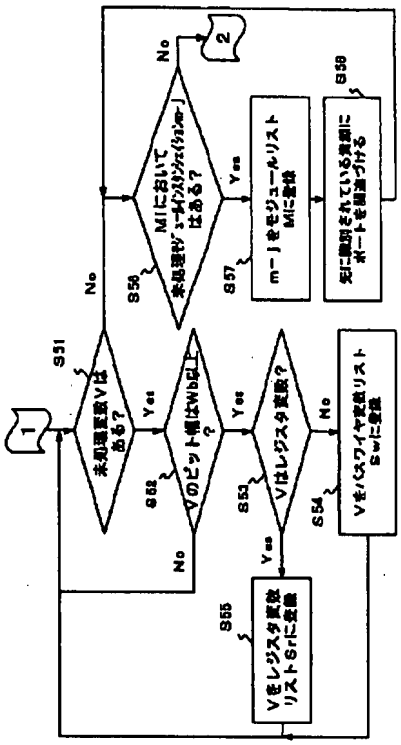
【図30】

最適化されたデータ転送グラフの例



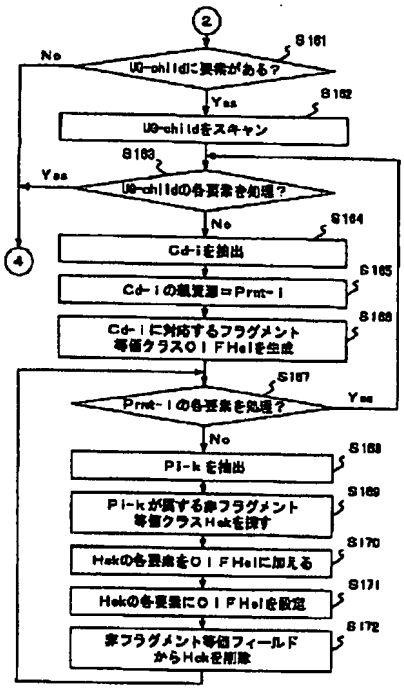
【図３３】

データ転送グラフを生成する方法のフローチャート(その2)



【図４１】

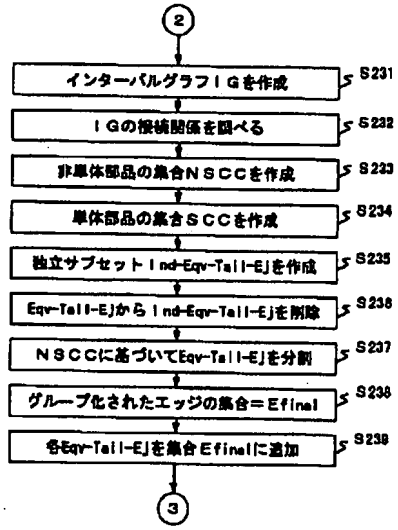
「フラグメント等価」を検出する処理のフローチャート（その４）



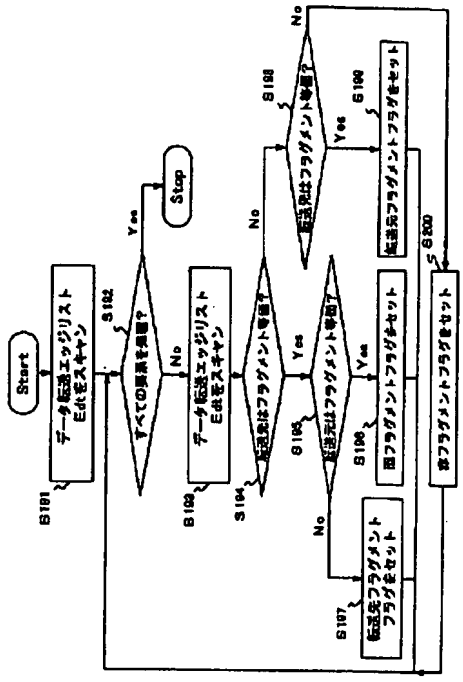
【図４３】

【図４５】

「等価エッジ」を検出する処理の
フローチャート（その２）

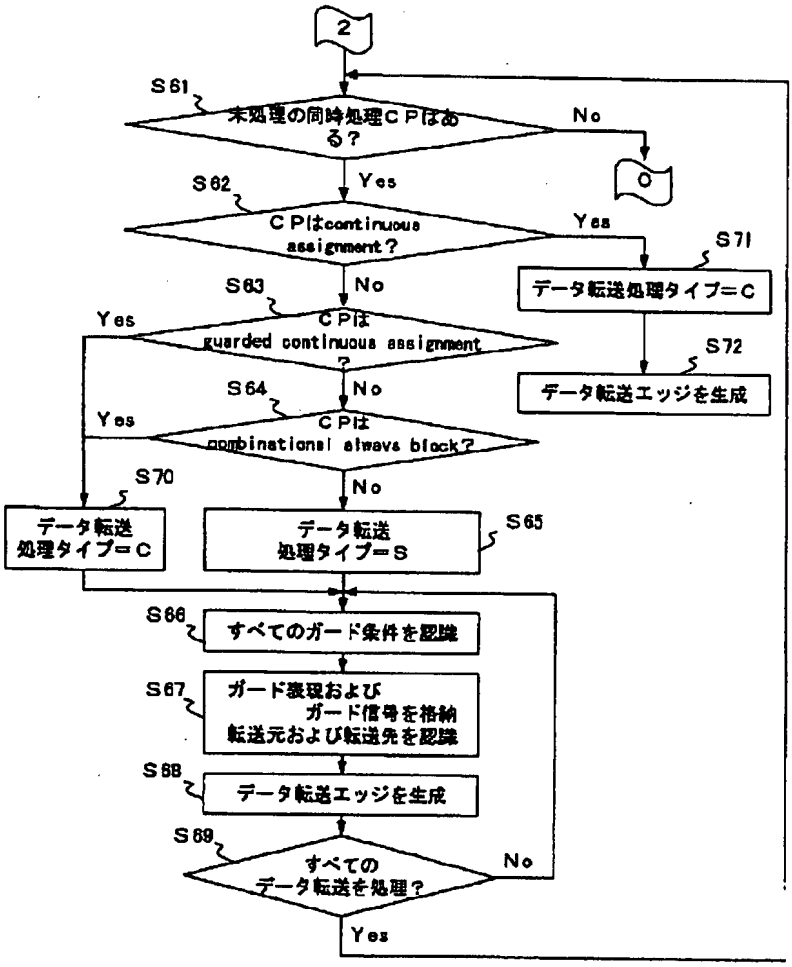


「分岐エッジ」を検出する処理のフローチャート



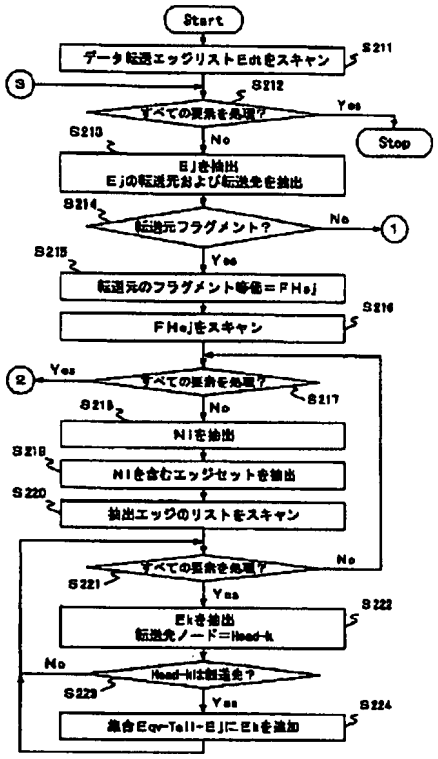
【図３４】

データ転送グラフを生成する方法のフローチャート（その３）



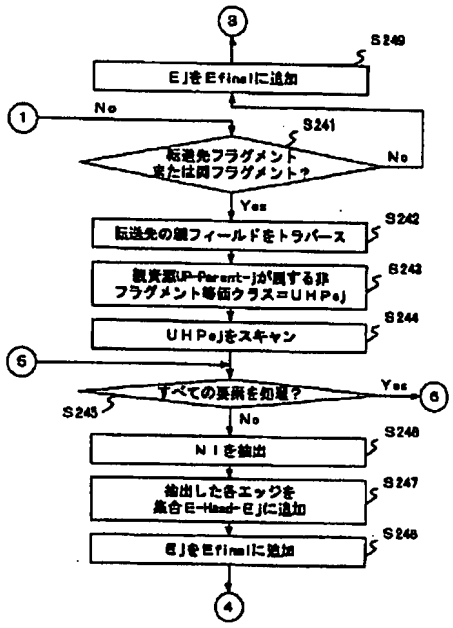
【図４４】

「等価エッジ」を抽出する処理のフローチャート（その１）



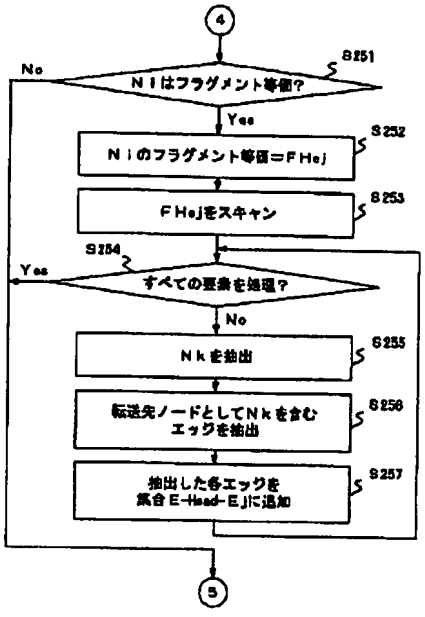
【図４６】

「等価エッジ」を検出する処理のフローチャート（その３）



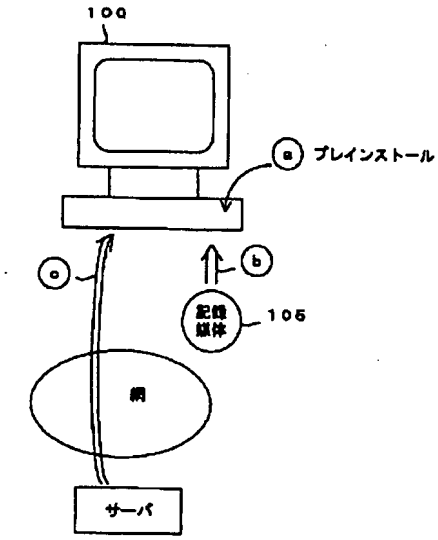
【図４７】

「等価エッジ」を検出する処理のフローチャート（その４）



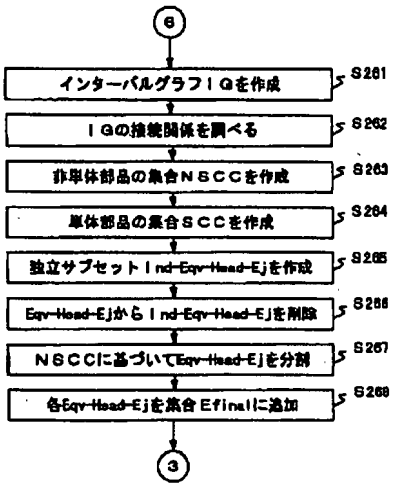
【図６２】

本発明に係わるソフトウェアプログラムなどの
提供方法を説明する図



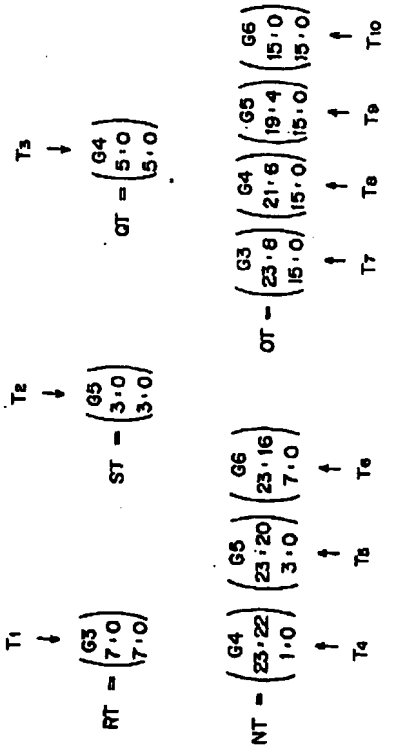
【図４８】

「等価エッジ」を検出する処理の
フローチャート（その５）



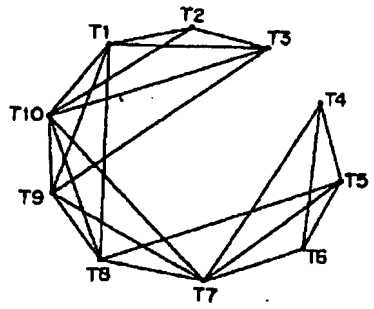
【図５０】

データ転送の定義



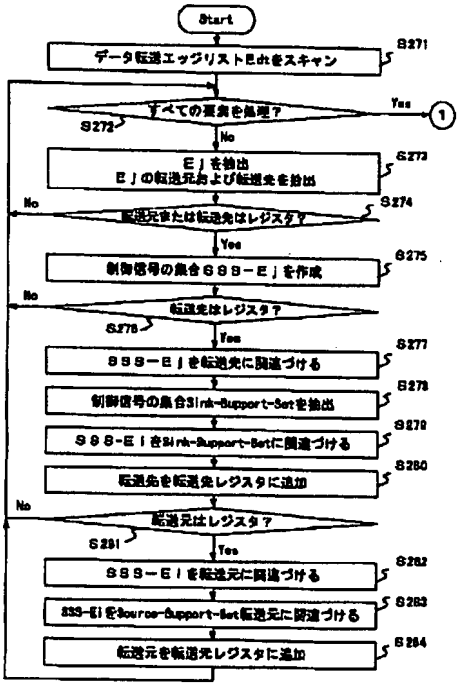
【図５１】

インターバルグラフの例



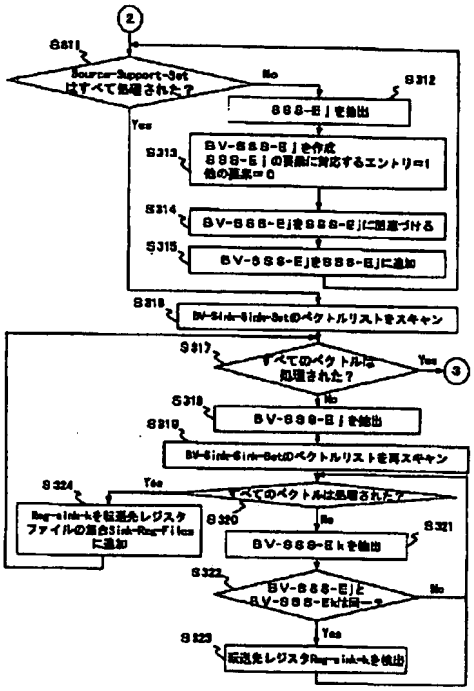
【図５２】

「レジスタファイル」を検出する処理のフローチャート（その１）



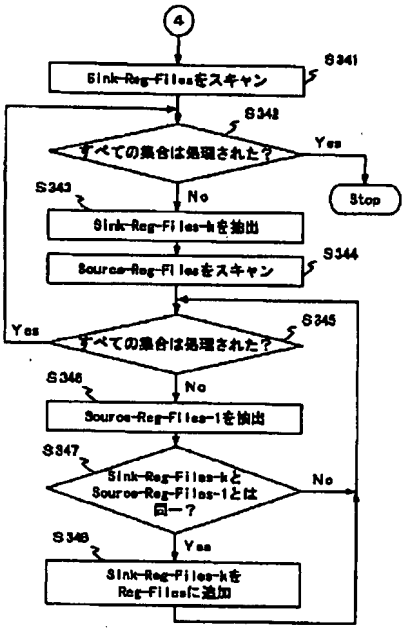
【図５４】

「レジスタファイル」を検出する処理のフローチャート（その３）



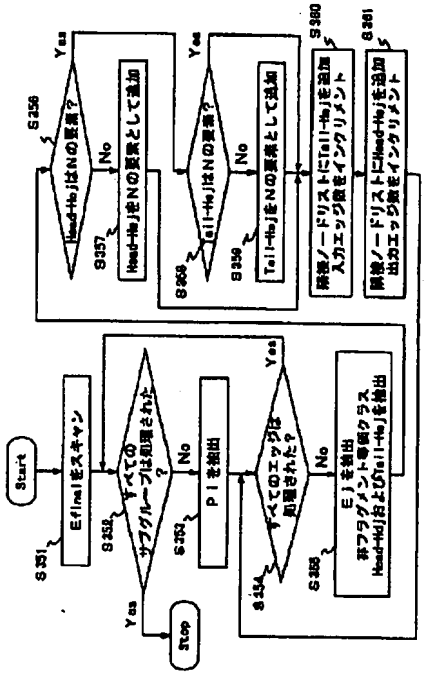
【図５６】

「レジスタファイル」を
検出する処理のフローチャート（その５）



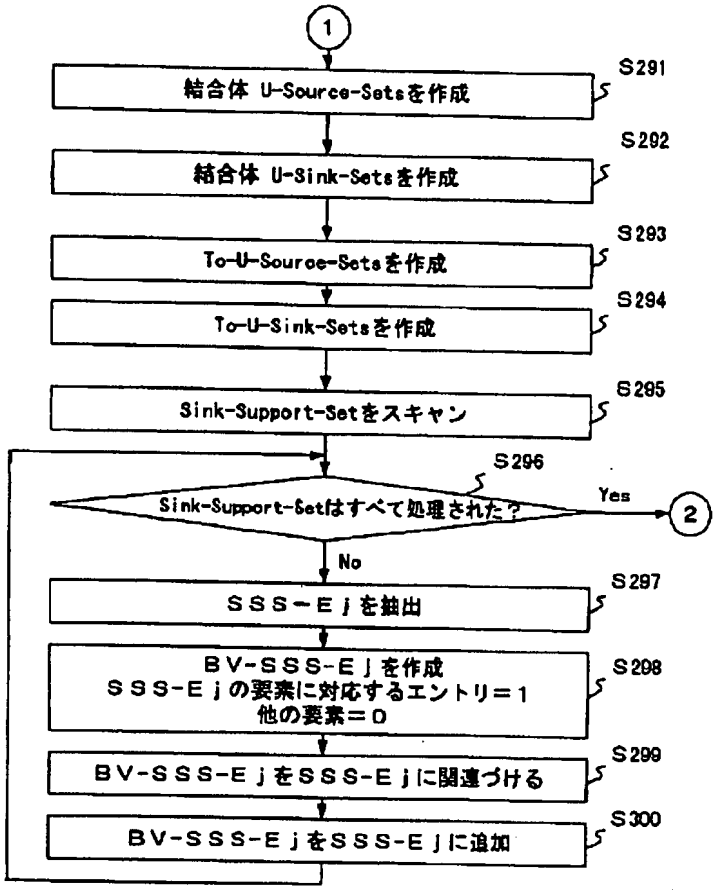
【図５７】

隣接ノードリストを作成する処理のフローチャート



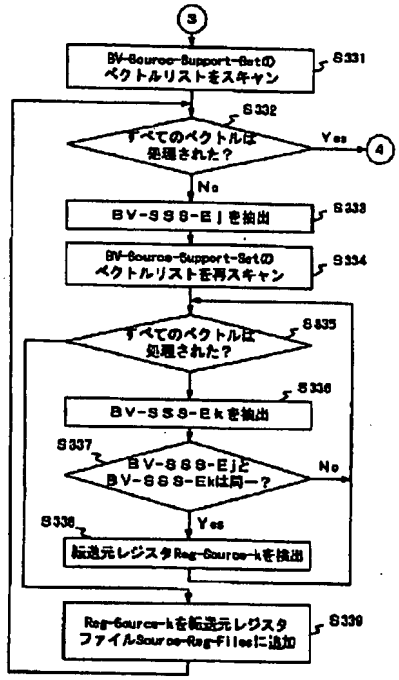
【図５３】

「レジスタファイル」を検出する処理のフローチャート（その２）



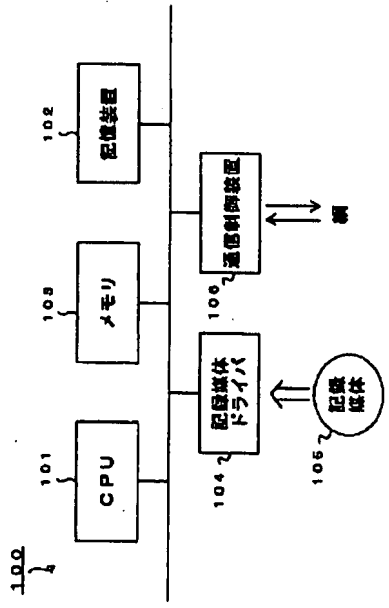
【図５５】

「レジスタファイル」を
検出する処理のフローチャート（その４）



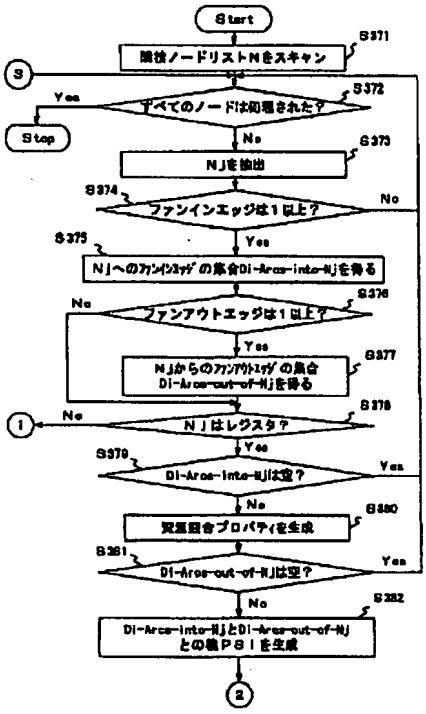
【図６１】

本発明の機能を記述した
プログラムを実行するコンピュータのブロック図



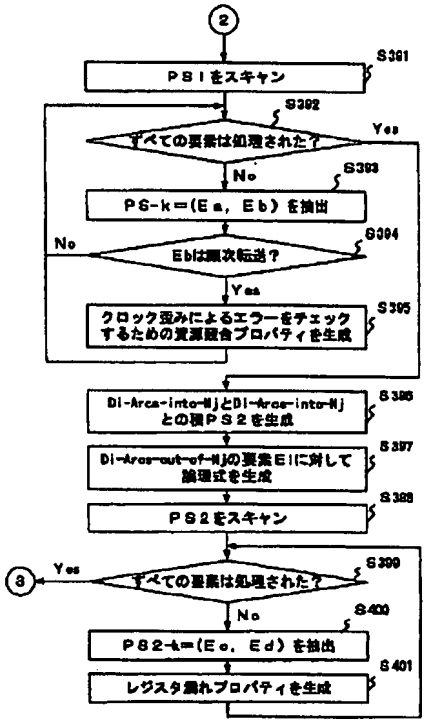
【図５８】

プロパティスクリプトを生成する処理のフローチャート（その１）



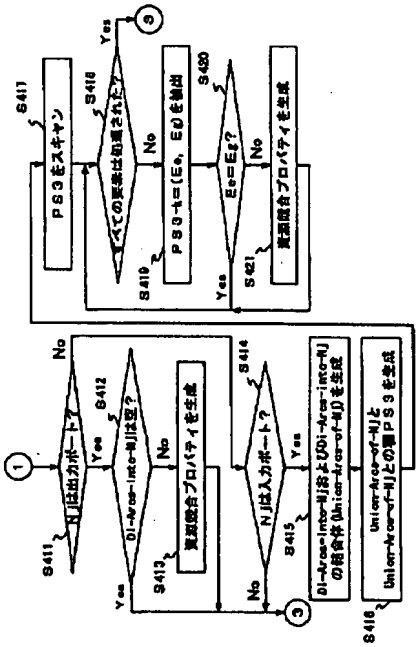
【図５９】

プロパティスクリプトを生成する処理のフローチャート（その２）



【図６０】

プロパティスクリプトを生成する処理の
フローチャート（その３）



フロントページの続き

(51)Int.Cl.'	識別記号	FI	テーマコード'(参考)
		H01L 27/04	U

(72)発明者　中田　恒夫
神奈川県川崎市中原区上小田中４丁目１番
１号　富士通株式会社内

Fターム(参考)　5B046 AA08 DA04 DA06 JA01
5F038 CA03 CA17 CD05 DF11 EZ10
EZ20
5F064 BB09 BB18 DD03 DD04 DD25
HH06 HH09 HH10 HH14